Heidelberg University
Institute of Computer Science
Database Research Group

Master Thesis

Background Linking of News Articles

Name:	Phi, Duc Anh
Matriculation Number:	3550091
Supervisor:	Prof. Dr. Michael Gertz
Submission Date:	Mai 28, 2021

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe und die Grundsätze und Empfehlungen “Verantwortung in der Wissenschaft” der Universität Heidelberg beachtet wurden.

Abgabedatum: Mai 28, 2021

Zusammenfassung

Das Ziel des *Background Linking von News Artikeln* ist es dem Leser zu helfen das Thema oder Kernaspekte eines gegebenen Nachrichtenartikels zu verstehen oder mehr darüber zu erfahren. Die Aufgabe ist es, bei einem gegebenen Nachrichtenartikel, andere Artikel zu finden, die dem Leser wichtigen Kontext oder Hintergrundinformationen liefern. In dieser Arbeit wird das *Background Linking* als Suchproblem behandelt, welches sich in zwei Schritte aufteilen lässt: Retrieval und Ranking. Basierend auf diesen zwei Schritten entwickeln wir ein Informationsgewinnungs-(IG-)Rahmenwerk in der Form einer konzeptionellen Suchmaschine. Diese kann als Leitlinie gesehen werden, da notwendige Komponenten für die Suche aufgezeigt werden, ohne spezifische Methoden oder Datenstrukturen vorauszusetzen. Das Rahmenwerk kommt mit technischen Überlegungen und unterstützt die Kombination von mehreren IG-Methoden. Basierend auf dem Rahmenwerk testen wir zwei drastisch unterschiedliche IG-Methoden anhand des *TREC Washington Post Korpus* und einem selbst gewonnenen Korpus, bestehend aus Nachrichten der Nachrichtenagentur *Netzpolitik.org*. Für letzteres konnten wir zeigen, dass interne Hyperlinks, welche vom Autor gesetzt wurden, als Heuristik für Background-Links genutzt werden können. Bei den besprochenen IG-Methoden handelt es sich um ein *Ranked-Boolean-Model*, welches uns als Basismethode dienen wird, und um eine *Semantische Suche* basierend auf *Satz-Embeddings* und *Hierarchical Navigable Small World (HNSW) Graphen* als zugrundeliegende Index-Struktur. Trotz optimierter *Satz-Embeddings* konnten wir zeigen, dass die *Semantische Suche* nicht effektiv ist für das Background Linking. Unsere Basismethode erzielt einen *nDCG@5*-Wert von 0.5205 auf den TREC 2020 Testdaten. Zum Vergleich: der höchste erreichte (zu diesem Zeitpunkt veröffentlichte) *nDCG@5*-Wert des *TREC News Track 2020* für das Background Linking liegt bei 0.5924. In diesem Kontext ist noch viel Verbesserungspotential für die Forschungsgemeinde vorhanden.

Abstract

The goal of the *Background Linking of News Articles* is to help a reader understand or learn more about the topic or main issues in the current article that they are reading. Given a news article, the task is to retrieve other news articles that provide important context or background information to the reader. In this work the background linking task is treated as a search problem that is divided into two steps: retrieval and ranking. Based on these two steps we develop an information retrieval (IR) framework in the form of a conceptual search pipeline that can be seen as a guideline, as it shows necessary components needed for search, without imposing specific methods or data structures. It comes with technical considerations and supports the use of multiple IR methods in parallel. On the basis of this framework we test two drastically different IR methods on the *TREC Washington Post Corpus* and on a self-scraped corpus, consisting of news articles from the German-speaking *Netzpolitik.org* news outlet. For the latter corpus we could show that internal hyperlinks placed by the author can be used as an heuristic for background links, which is effective for evaluating retrieval recall. The bespoke IR methods are the *ranked Boolean model*, which serves as a baseline method, and a *semantic search* approach based on *sentence-embeddings* and *Hierarchical Navigable Small World* (HNSW) graphs as the underlying index structure. Even with optimized sentence-embeddings, we could show that the semantic search approach is not effective for both the retrieval and ranking task, not even as a complementary approach, as a better recall-to-precision ratio can be achieved for the baseline method. Our baseline approach achieves an *nDCG@5* score of 0.5205 on the TREC 2020 test data. In comparison the best performing model (that was published at the time of this writing) among all *TREC News Track 2020* submissions for the background linking task achieves an *nDCG@5* score of 0.5924. In this context there is still room for improvement for the research community.

Acknowledgements

First and foremost I would like to thank Prof. Gertz for his supervision and constant support. His constructive criticism and encouraging manner helped me tremendously during the writing of this work. I could not have wished for a better supervisor.

I also want to thank my family and friends supporting me in every way.

I am grateful. Thank you.

Contents

1 Introduction

In the digital news domain the background linking task is about finding other news articles, given a current article, that support the reader's understanding by providing important context or background information.

"As news consumption has moved online, an increasing portion of the U.S. population is turning away from traditional news sources such as radio and newspapers and relying on digital platforms to keep up to date" [1].

"According to Pew Research studies, in 2016, roughly 38%*" of Americans got their news online, with the fractions increasing for younger consumers, and in 2018* 93% *of American adults get at least some of their news online"* [2].

It is apparent that background linking becomes increasingly important, helping individuals to stay informed. However, solving this problem can have a vastly far-reaching impact than facilitating news consumption, as the problem of background linking can be extended to any domain. Suppose in a legal domain you are working on a case and try to find similar cases in the past or get more related background information. The legal office has an archive of all the data you need, but it is tedious to find the right information and you might miss out on important relevant documents when going through the archive manually.
The stated problem is very similar to the background linking task of news articles and only differs in its domain and definition of relevance. In this context, it makes sense to limit the scope to a single domain first and achieve good results before expanding the background linking task across multiple domains. This way, domain-specific problems can be identified. Thus, we are going to focus on the news domain in this work.

1.1 Problem Setting and Objectives

In essence, what we are trying to achieve in this work is to make large collections of documents as accessible as possible to gain valuable insights. What we mean by ”as accessible as possible” is to satisfy an information need from a reader, without a reader’s active intervention. Like a recommendation system, ”relevant” documents should be displayed in an ordered manner, such that the most relevant ones are on top. Now there is the question of how to define relevance and how can it be measured, such that documents can be compared and ordered based on that metric? In this context, relevance is an ambiguous and complex concept that is highly domain-specific and cannot be easily mapped to an algorithm.

”[...] the notion of background relevance is not well-studied and it requires better understanding to ensure effective system performance” [3].

The best assessors of relevance are obviously humans. Thus, human-rated relevance judgments are seen as a ground-truth, even though various judgments might correspond to different (subjective) tastes. For this reason, human-rated judgments are essential for evaluating background linking models. Unfortunately, human-rated data are generally sparse, due to their labor-intensive (and thus expensive) nature. So there is an additional need for finding alternative heuristics or methods for labeling/rating relevant documents.
In contrast to Google and other search engines it is not feasible to analyze user data and user behavior in order to improve relevant results. The information available for the background linking task is limited to the textual content of corresponding documents. This is in line with the information presented to a human rater. In this context the objective of this work can be phrased as extracting relevance signals from a document’s textual content that can be used to model human-rated relevance judgments, and to rank documents accordingly.
Furthermore, there is the problem of handling large document collections. Given a query document, how do we find the most relevant documents in a collection of potentially millions of documents in near real-time? There is the need to find appropriate data structures, such that rapid and accurate retrieval of documents

is possible, with respect to a query. This is important, as users expect recommendations or search results at an instant, especially in the age of Google and social media.
The stated problems that correspond to the background linking task fall into the scope of *information retrieval*, that is, an academic field of study:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" [4, p.1].

1.2 Contributions

In this work we address the background linking task in the news domain and make the following contributions:

- We propose a general IR framework in the form of a conceptual search pipeline, which is a promising model for solving the background linking task. The idea is to deploy and combine multiple IR methods in parallel that could yield diverse retrieval results a single-model system might miss. Furthermore, multiple IR methods introduce several features that could be beneficial for the ranking task.

- We develop a *semantic search* method that is based on the state-of-the-art sentence embedding model and an efficient data structure. By applying semantic search on labeled corpora we found out that sentence embeddings, as the unit of search, are not effective for the background linking, as they are outperformed by traditional IR methods. We come to this conclusion after experimenting with sentence embeddings that are derived from different parts and combinations of text in the underlying document.

- To have more diversity in our data, we generate a German corpus of IT-related news (from *netzpolitik.org*) and propose a method to automatically label documents based on a developed heuristic. Our experiments show similar retrieval results on these automatically labeled documents in comparison to a different human-labeled document collection.
- We open-source the source code of this project on Github:

  https://github.com/DucAnhPhi/NewsSearchEngine

  The repository contains (installation) instructions and executable scripts, such that all conducted experiments, methods and results that are presented in this work can be reproduced or built up on.

1.3 Outline

Chapter 2 introduces relevant fundamentals and covers related work in the context of background linking. The following Chapter 3 builds up on the fundamentals and discusses how the background linking task can be solved. An IR framework in the form of a conceptual search pipeline is subsequently proposed. Next, in Chapter 4 the bespoke framework and considered approaches are implemented. Furthermore, experiments are conducted to evaluate the efficacy of corresponding methods. Finally, Chapter 5 gives a summary of this work and the results are discussed with regard to future work.

2 Fundamentals

This chapter introduces core concepts that are referred to in this work and are essential for the understanding. Initially, Section 2.1 introduces concepts of natural language processing (NLP), starting from the basics, such as text normalization and keyword extraction, leading up to machine learning concepts such as transformers and sentence embeddings. In the subsequent Section 2.2 we introduce concepts and methods that are related to the research area of information retrieval (IR), e.g., the inverted index, the Okapi BM25 measure, and more. Finally, we cover related work in the context of the background linking task.

2.1 Natural Language Processing

In this work we are dealing with textual data, more specifically natural language, which in contrast to machine-readable languages (such as programming languages) is highly unstructured, ambiguous and arguably much more complex. In order to process and manage natural language text we make use of techniques belonging to the field of **natural language processing** (NLP).

"NLP is defined as a specialized field of computer science and engineering and artificial intelligence with roots in computational linguistics. It is primarily concerned with designing and building applications and systems that enable interaction between machines and natural languages evolved for use by humans" [5, p.46].

2.1.1 Text Normalization

Text normalization is defined as a process that consists of a series of steps that should be followed to clean and standardize textual data [5, p.115].

These steps are covered in the following.

Text Tokenization: In the context of NLP *"tokens are independent and minimal textual components that have some definite syntax and semantics"* [5, p.108]. Tokenization is the process of splitting a string of text into tokens. Tokens can have various granularity levels, e.g., a text can be broken down into sentences that can be further broken down into clauses, phrases and words - which all correspond to tokens [5, p.108]. We are mainly interested in word tokenization, which is important for processes like cleaning and normalizing text. A simple example for a word tokenizer can be seen in Figure 2.1. It splits a string into words, based on whitespaces.

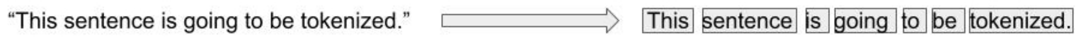


Figure 2.1: Whitespace Tokenizer

Removing Special Characters: Usually punctuation and special characters are not relevant for analyzing or extracting information from text and should be removed. However, for some domains or problems special characters or punctuation could be relevant, e.g., link addresses or mentions denoted with *"@"*. Thus the removal of special characters should be done with care.

Removing Stopwords: Stopwords are words that have little to no significance. They are usually removed from text to retain words having maximum significance and context. Exemplary stopwords are *"the"*, *"a"*, *"and"*, etc. There is no universal or exhaustive list of stopwords. Each domain or language might have its own set of stopwords. [5, p.120]

Lowercasing: Words with different casing should be considered the same, thus all words should be converted to lowercase.

Stemming/Lemmatization: Word stems are often known as the base form of a word [5, p.128]. Consider the words *"happiness"* and *"happier"*. Both words can be *stemmed* to *"happi"*, thus considering both as the same word. Notice that *"happi"* is not a real English word, which, depending on the context, may not matter. *Lemmatization* is similar to stemming, but converts words to their root word, namely *lemma*, instead of their root stem [5, p.131]. Lemmas are real words that can be found in a dictionary, in contrast to root stems. Thus, both words *"happiness"* and *"happier"* can be lemmatized to *"happy"*. The objective of stemming/lemmatization is to be invariant to word inflections, such that there is no distinction between words that share the same root word/stem.

2.1.2 Keyword Extraction

Keywords can be defined as a sequence of one or more words that ideally represent the essential content of a document in condensed form, and are widely used in information retrieval systems [6, p.3]. The aim of *keyword extraction* is to find a small set of terms that describe a specific document [7]. The two most common automatic keyword extraction techniques are described in the following:

TF-IDF is a metric widely used in information retrieval and stands for a combination of two metrics: *term frequency (TF)* and *inverse document frequency (IDF)* [5, p.181]. This measure assumes a collection of documents $D = \{d_1, d_2, \ldots, d_n\}$. The idea of keyword extraction with *TF-IDF* is to compute this measure for each word in a document $d_i \in D$. Subsequently, the highest scoring k words are considered as the document's keywords. Suppose tf_{w,d_i} denotes the frequency of a word w in document d_i. Furthermore, suppose that df_w denotes the document frequency, that is, the number documents $d_i \in D$ in which the word w is present. Then, the inverse document frequency is defined as $idf_w = 1 + ln\left(\frac{|D|}{df_w+1}\right)$. Finally, TF-IDF for a word w in document $d_i \in D$ is defined as $tfidf_{w,d_i} = tf_{w,d_i} \cdot idf_w$ [5, p.182].

RAKE stands for *Rapid Automatic Keyword Extraction* and is a method for extracting keywords from individual documents, in contrast to TF-IDF that assumes a collection of documents. The *RAKE* algorithm requires a list of stopwords and a set of phrase- and word-delimiters. The idea is to split the document into candidate words based on aforementioned stopwords and delimiters. After every candidate keyword is identified, a graph of word co-occurrences is build. Based on that graph, word scores are calculated that are defined as the degree of a word divided by its frequency. Finally, the highest scoring k candidate words are considered as the document's keywords [8].

2.1.3 Introduction to Supervised Machine Learning

Supervised machine learning can be seen as an approximation of an *ideal* function. Generally speaking, a function typically receives an input and provides an output based on the given input. An ideal function provides the expected output to any given input. In this context, the notion of *"supervised"* refers to having an idea of what the expected output should look like.

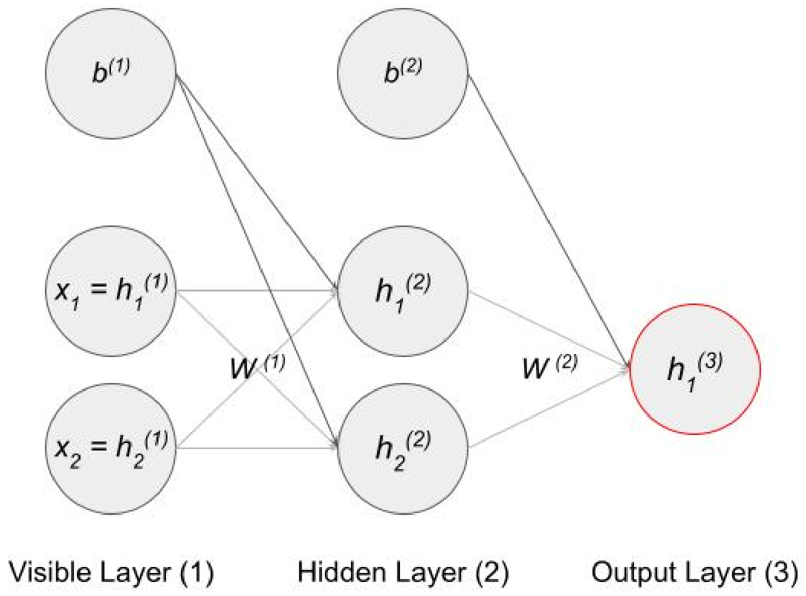


Figure 2.2: Fully connected feed-forward network

A function can be represented in many ways, one of which is a (multi-layered) *artificial neural network* that is shown in Figure 2.2. To be more precise, the shown network is a *feed-forward* network that is characterized by its *unidirectional information flow.* An artifical neural network is essentially a series of linear and non-linear combinations of the input data X, bias units b, and weights W, to compute a prediction y^*. Computing a prediction for a feed-forward network is called a *forward pass* and is calculated as the following, with respect to Figure 2.2:

$$y^* = h_1^{(3)} = g\left(H^{(2)} W^{(2)\,T} + b^{(2)}\right)$$
$$H^{(2)} = g\left(H^{(1)} W^{(1)\,T} + b^{(1)}\right)$$
$$H^{(1)} = X$$

where $H^{(l)}$ corresponds to the matrix representation of all (hidden) values in layer l, $W^{(l)}$ is the weight matrix for layer l, g is the *activation function*, and X is the matrix representation of the input data, that is, x_1 and x_2.
The activation function is a non-linear function that introduces non-linearity to the model. This way, the network is able to approximate more sophisticated functions that are not only linear combinations of its inputs. A common activation function is the *Softmax* function, which turns a vector $\vec{v}$ of K real values v_k into a vector of K real, positive values (between 0 and 1) that sum up to 1 [9]:

$$softmax\left(\vec{v}\right)_j = \frac{e^{v_j}}{\sum_{k=1}^{K} e^{v_k}}$$

As mentioned earlier the objective of supervised machine learning is to approximate an ideal function, thus we need to know how close our model's predictions come to the expected outputs. This is measured by a *cost function*, which is also referred to as an *objective function.* We cover two commonly used objective functions in the following:

- **Mean squared error** (MSE) is defined as:

$$L_{MSE} = \frac{1}{N}\left[\sum_{i=1}^{N}\left(y_i^* - y_i\right)^2\right]$$

  where y_i^* and y_i are the predicted and expected values respectively for observation i.

- **Cross-entropy loss** or *log loss* measures the performance of a classification model whose output is a value between 0 and 1, and is defined for classes $C = 2$ as [10]:

  $$L_{log} = -\tfrac{1}{N} \textstyle\sum_{i=1}^{N} \left[y_i \cdot log\left(y_i^*\right) + (1 - y_i) \cdot log\left(1 - y_i^*\right)\right]$$

  And if $C > 2$ the cross-entropy loss is defined as:

  $$L_{log} = -\tfrac{1}{N} \textstyle\sum_{i=1}^{N} \left[\sum_{c=1}^{C} y_{i,c} \cdot log\left(y_{i,c}^*\right)\right]$$

  where y is a binary indicator (0 or 1) if class label c is the correct classification for observation i.

It is desired that our model's loss that results from a corresponding objective function, is at a minimum. As the objective function is not totally differentiable, it is improved step-by-step in rounds or *iterations*. This process is referred to as *training* or *learning*. But how do we iteratively change the model's weights to find its minimum? Generally speaking, the local gradient ΔL of a function points away from a local minimum. Thus, to find a function's local minimum one has to follow the opposite direction of the gradient, that is, changing the function's weights W accordingly:

$$W = W - \alpha \Delta L$$

where α denotes the *learning rate* that weights the gradient and can be seen as the "step size" we take towards a local minimum. This method is known as *gradient descent*. The gradient ΔL for a objective function $L(W)$ with corresponding weights W is the partial derivative of the function with respect to W:

$$\Delta L = \tfrac{\delta L(W)}{\delta W}$$

2.1.4 Embeddings

Generally speaking, *embeddings* are low-dimensional representations of their original data in the form of a fixed array of numbers, that is, *vectors* (embeddings and vectors are used synonymously from now on).

Embeddings share a *latent vector space*, which is a representation of data, where data points that are "close" to each other exhibit similar semantics and vice versa. In this work we mainly focus on *sentence embeddings* that represent the semantics of their textual counterpart.

Comparing Sentence Embeddings As vectors can be mathematically compared to each other, one is able to compare semantical meanings. So the question is how do we quantify semantic similarity when comparing two vectors? The Euclidean distance might come to mind first. However, it is not suitable for this task, as it incorporates absolute term frequencies. This way, two semantically similar documents could have a significant Euclidean distance, simply because one is much longer than the other [4, p.121].

"To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the ***cosine similarity*** *of their vector representations $v(d_1)$ and $v(d_2)$"* [4, p.121]:

$$cossim\,(d_1, d_2) = \frac{\vec{v}(d_1)\cdot\vec{v}(d_2)}{||\vec{v}(d_1)||\cdot||\vec{v}(d_2)||}$$

The numerator represents the dot product of both vectors, while the denominator is the product of their Euclidean lengths [4, p.121].

2.1.5 Transformer

A *Transformer* is a network structure, more specifically an *encoder-decoder architecture* that is based solely on *attention mechanisms* [12].

Self-attention The idea behind *self-attention* is to incorporate the "understanding" of other relevant words into the one that is currently processed (see Figure 2.3) [11]. The self-attention calculation is based on three matrices that are denoted as Q, K and V, which the authors refer to as *queries*, *keys* and *values* respectively. These matrices are calculated by multiplying a matrix X that corresponds to the input embeddings packed together by trained weight matrices W^Q, W^K and W^V respectively (see Figure 2.4).

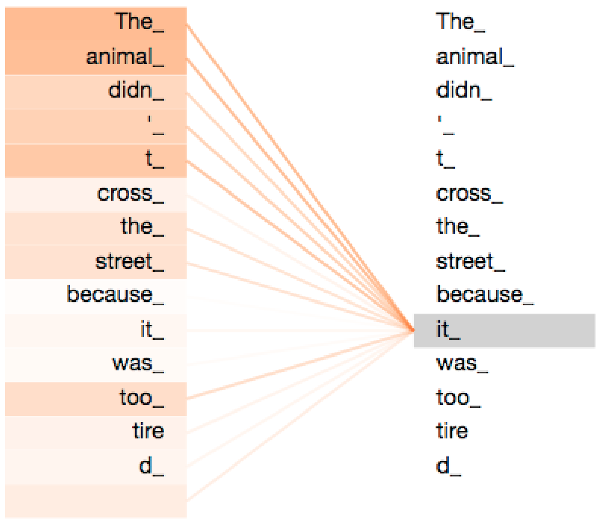


Figure 2.3: Visualized self-attention mechanism for encoding the word "it". Self-attention allows the word "it" to be associated with "animal". The Figure was taken from [11].

The self-attention embeddings are then calculated as the following [12]:

$$Attention\left(Q,K,V\right) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

with d_k denoting the number of dimensions for Q and K. In the paper this is referred to as *Scaled Dot-Product Attention*.
Instead of performing a single attention function, the authors found it beneficial to perform the attention function $h = 8$ times in parallel, with different, learned weight matrices W_i^Q, W_i^K and W_i^V for $i \in \{1, 2, \ldots, h\}$. The resulting h self-attention matrices are concatenated and once again projected, resulting in a final self-attention matrix [12]:

$$\begin{gathered} MultiHead\left(Q,K,V\right) = Concat\left(head_1, \ldots, head_h\right)W^O \\ \text{where } head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{gathered}$$

In the paper this process is referred to as *multi-head attention* that *"allows the model to jointly attend to information from different representation subspaces at different positions"* [12].

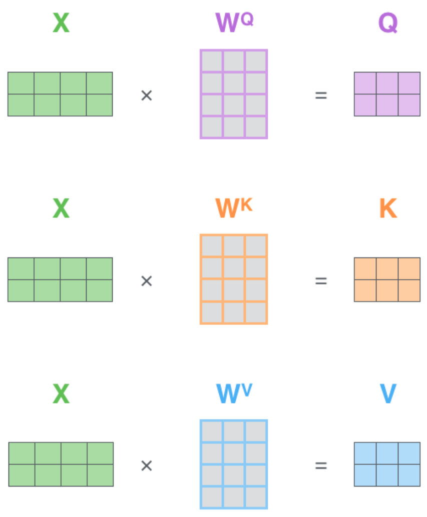


Figure 2.4: Self-attention calculation: query, key and value matrices. The Figure was taken from [11].

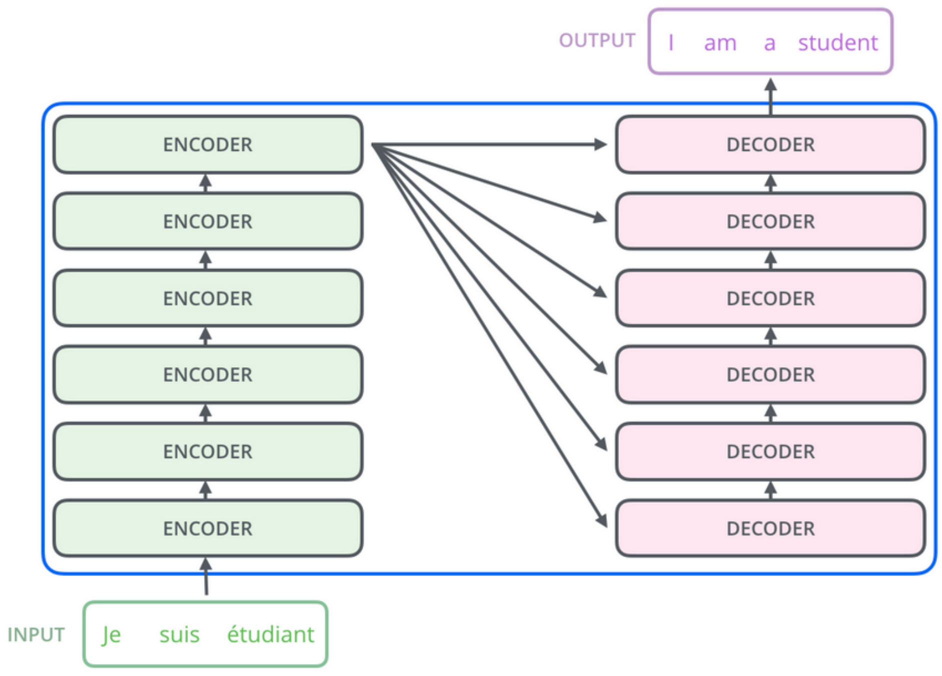


Figure 2.5: Simplified Transformer architecture with encoder and decoder stacks. The Figure was taken from [11].

Model architecture A simplified Transformer architecture with the encoder and decoder component are shown in the left and right halves of Figure 2.5 respectively. Both components are composed of a stack of $N = 6$ identical layers.

"Here, the encoder maps an input sequence of symbol represenations $(x_1, \ldots, x_n)$ *to a sequence of continuous representations* $z = (z_1, \ldots, z_n)$*. Given* z*, the decoder then generates an ouput sequence* $(y_1, \ldots, y_m)$ *of symbols one element at a time"* [12].

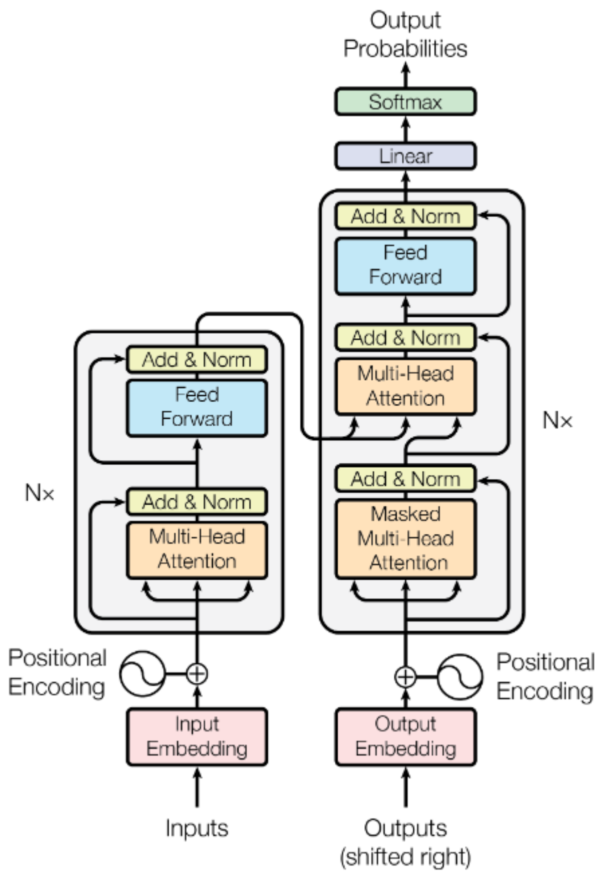


Figure 2.6: The Transformer - model architecture [12, Figure 1].

Encoder The encoder is composed of two sub-layers: a multi-head self-attention mechanism and a fully connected feed-forward network (FFN) (see Section 2.1.3) that is applied to each position separately and identically (see Figure 2.6) [12]:

$$FFN(X) = max\left(0, XW_1 + b_1\right)W_2 + b_2$$

While the linear transformations (for the FFN) are identical across different positions, they use different parameters from layer to layer.
Furthermore, the authors employ a *"residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is $LayerNorm\left(x + Sublayer\left(x\right)\right)$, where $Sublayer\left(x\right)$ is the function implemented by the sub-layer itself"* [12]. Layer normalization is a technique developed by Ba et al., which normalizes the summed input of a layer to reduce the training time of the underlying model [13].

Decoder The decoder's structure is similar to the encoder, but has the following modifications:

- There is a third sub-layer referred to as *"encoder-decoder attention"* layer that performs multi-head attention over [12]:
  - The output of the encoder stack, that is, the output embeddings of the last encoder layer that were transformed into a set of attention vectors K and V [11].
  - The queries Q from the previous decoder layer.
- The decoder works in a sequential manner, generating tokens one element at a time. At each step the model is *auto-regressive*, meaning it consumes the previously generated tokens as additional input when generating the next. To preserve this auto-regressive property, the self-attention sub-layers are only able to attend from previous positions up to the current one in the output sequence. This is achieved by "masking" future positions, that is, setting them to negative infinity, before the softmax step in the self-attention calculation [11]. This modified attention layer is referred to as *Masked Multi-Head Attention* (see Figure 2.6).

Additionally, the decoder has a final *Linear* and *Softmax* layer in order to turn embeddings into words. The Linear layer is a fully connected feed-forward network that projects the output embedding (produced by the decoder stack) into a much larger, so-called *logits vector*. This vector has the size of the output vocabulary, where each dimension corresponds to an unique English word [11]. The Softmax layer subsequently maps respective values to values between 0 and 1 that correspond to probabilities (see Section 2.1.3). Finally, the highest probability is chosen, and the word associated with it is the output for the current time step [11].

Input and output embedding Both the decoder and the encoder stacks assume embeddings as their input (see Figure 2.6). For this, the authors use *"learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}"* [12].

Positional encoding In order for the model to make use of the order of the sequence, information about a token's position in the sequence is injected in the form of *"positional encodings"* at the start of the encoder and decoder stacks (see Figure 2.6) [12]. As the positional encodings have the same dimension size like the embeddings, both can be simply summed.

2.1.6 Bidirectional Encoder Representations from Transformers (BERT)

Model BERT is a language model that is divided into two steps of how it is developed [14]:

1. **Pre-training:** During pre-training, the model is trained on unlabeled data, that is, large amounts of text, over two tasks [15]:
    - **Task 1: Masked Language Modeling (MLM)** Standard language models can only predict the next word given previous words, from left-to-right or right-to-left. Thus, they only look into one direction. However, it is preferable to look into both directions in order to consider

the left and right context of a target word [16]. The MLM task enables the model to look at both directions, which Devlin et al. refer to as *bidirectional conditioning*. This way, the model can *"trivially predict the target word in a multi-layered context"* [15]. For this task 15% of the input tokens are selected at random. A selected token is then:

a) masked 80% of the time,

b) randomly replaced with another token 10% of the time or

c) is unchanged 10% of the time.

The model should subsequently predict the correct token at the chosen token position. The cross entropy loss function is optimized for the predictions.

- **Task 2: Next Sentence Prediction (NSP)** The NSP task is necessary for the model to understand relationships between sentences. Given two sentences A and B the model should predict whether B follows A. For the training data, 50% of the time B is the actual next sentence that follows A, and 50% it is a random sentence from the training corpus.

2. **Fine-tuning** refers to the small changes to the BERT model after the pre-training phase. The model is initialized with the pre-trained parameters and *"can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications"* [15]. In Section 2.1.7 we show how the BERT model can be fine-tuned to create meaningful sentence embeddings.

Architecture BERT's architecture is based on Transformer encoder layers (see previous Section 2.1.5) that are stacked on top of each other [14] (see Figure 2.7).

Input BERT represents its input as a sequence of tokens, which may refer to a single or two sentences. A "sentence" is seen as an "arbitrary span of contiguous text, rather than an actual linguistic sentence" [15].

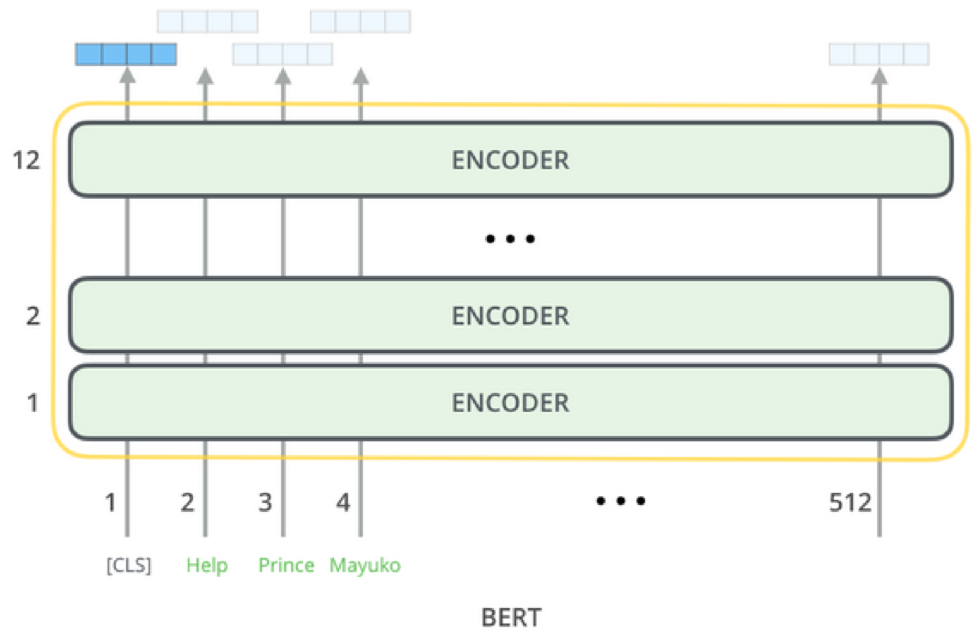


Figure 2.7: BERT architecture based on Transformer encoder layers. The Figure was taken from [14].

Initially, the input text is tokenized into chunks rather than words. Furthermore, additional tokens are added at the start and at the end of the tokenized sentence [17]:

- At the start a special classification token ([CLS]) is added that serves as an input representation for classification tasks.
- At the end a special separation token ([SEP]) is added that marks the end of a sentence and separates sentence pairs.

Each input token needs to be represented as an embedding before it can be passed to the first encoding layer of the BERT model (see Figure 2.7).

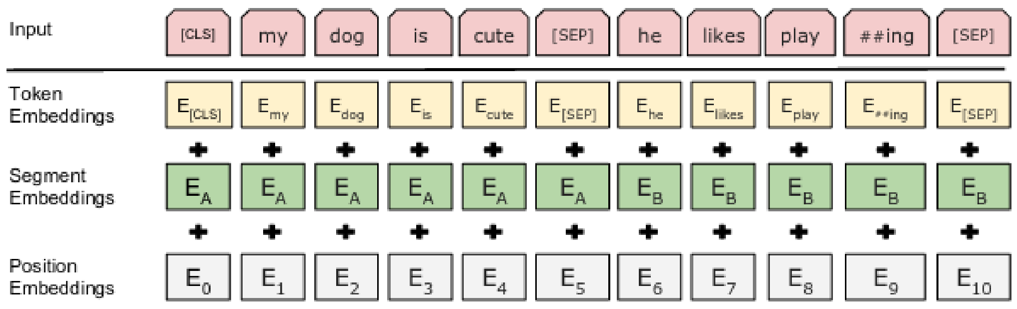


Figure 2.8: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings [15, Figure 2]

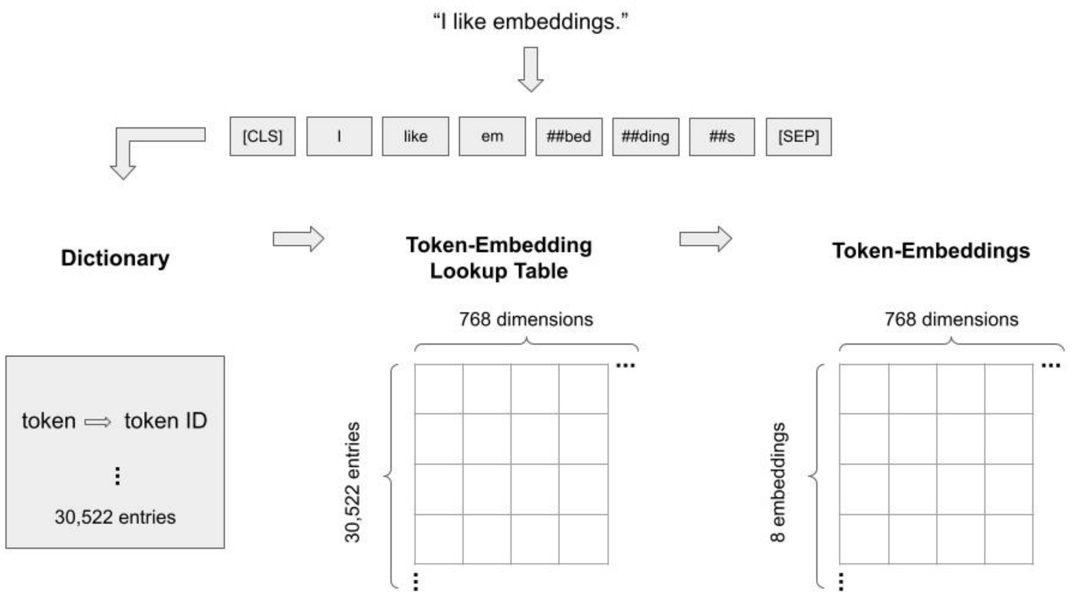


Figure 2.9: BERT: Process of computing token embeddings for the input.

Figure 2.8 shows how input embeddings are calculated, that is, a sum of three corresponding embeddings:

- **Token embeddings:** BERT deploys *WordPiece* embeddings that were developed by Wu et al. [18], which enables BERT to represent all English words as embeddings with a relatively small vocabulary size of $30,522$.

The idea behind WordPiece embeddings is to split words into word pieces that can be reused and combined to represent other words. A word piece can correspond to a full word, a substring or, at its smallest form, a single character (the latter two are marked with a prefix of two hashtags). These word pieces are then projected to the same latent space with 768 dimensions. To calculate a token embedding is to simply search through a dictionary to find the corresponding token ID, which is subsequently used to find the respective token embedding in a lookup table that represents BERT's vocabulary (see Figure 2.9).

- **Segment embeddings:** For the aforementioned NSP task a pair of input sentences are concatenated and fed into the model [17]. To distinguish both sentences the authors make use of *segment embeddings* that encode the belonging of an input token to a respective sentence. Thus, there are only two possible segment embeddings. The process of computing segment embeddings is to simply assign the corresponding segment embedding to a token, depending on its sentence belonging (see Figure 2.10).
- **Position embeddings:** BERT consists of a stack of Transformer Encoders, that require position embeddings to preserve the sequential nature of the input [17]. As BERT supports input sequences of up to a length of 512, the authors trained BERT to learn 512 vector representations that correspond to each position, with a dimension of 768 [17].

Output Each input token outputs a respective *"final hidden state"* [15] with the dimension size $H = 768$. The final embedding of the [CLS] token is denoted as $C \in \mathbb{R}^H$ and the final hidden vector for the i^{th} input token is denoted as $T_i \in \mathbb{R}^H$ [15]. Vector C can be used for classification tasks and achieves great results with just a single-layer neural network as the classifier [14].

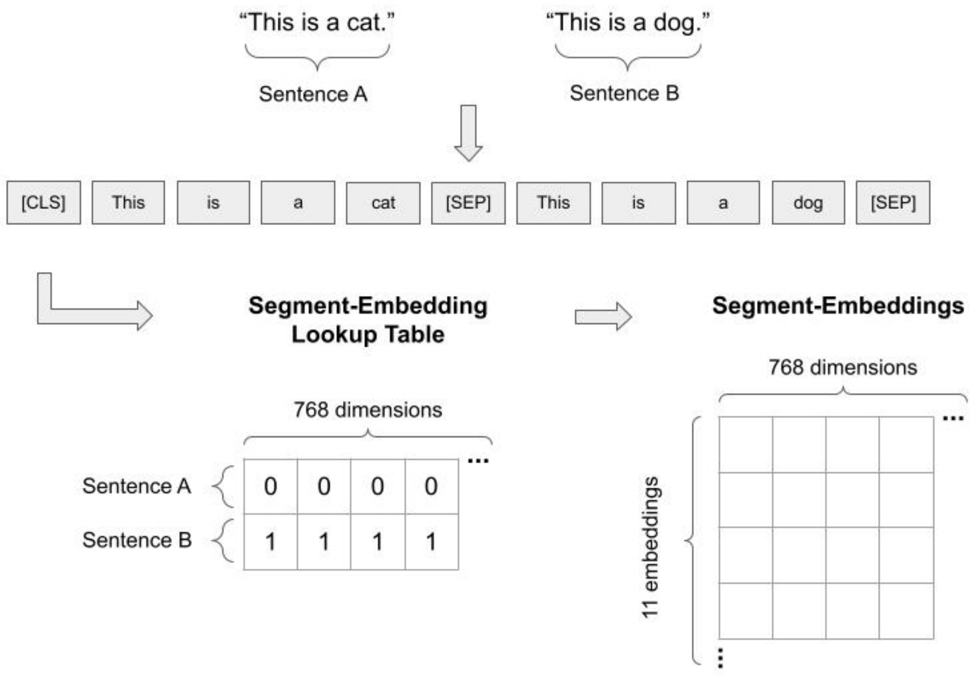


Figure 2.10: BERT: Process of computing segment embeddings for the input.

2.1.7 Sentence-BERT (SBERT)

S-BERT is a modification of the BERT network (see previous Section 2.1.6) that is able to derive semantically meaningful sentence embeddings [19]. For the modification, Reimers et al. [19] add a mean-pooling layer for the output embeddings of the BERT network, that is, a mean of all output vectors is computed to derive a fixed size sentence embedding. The authors fine-tune the underlying BERT model by creating *"siamese and triplet networks to update the weights such that the produced sentence embeddings are semantically meaningful and can be compared with cosine similarity"*.
For the fine-tuning, Reimers et al. choose between three objective functions with corresponding network structures depending on the available training data [19]:

- **Classification Objective Function:** Sentence embeddings u and v are concatenated with the element-wise difference $|u - v|$ and multiplied by the trainable weight $W_t \in \mathbb{R}^{3n \times k}$, where n denotes the dimension size of the sentence embeddings and k denotes the number of classes.

The *Softmax function* (see Section 2.1.3) is then applied. Thus, the output of the corresponding model, that is, a predicted probability p that an observation o is of class c, is defined as:

$$p_{o,c} = softmax\left(W_t\left(u, v, |u - v|\right)\right)$$

The classification objective function is based on the cross-entropy loss (see Section 2.1.3). The classification objective function is deployed for labeled training data, such as the SNLI collection, which consists of $570,000$ sentence pairs that are annotated with the labels *contradiction*, *entailment*, and *neutral*. The classification objective function with its underlying structure is shown in Figure 2.11.

- **Regression Objective Function:** The cosine-similarity between two sentence embeddings u and v is computed for the output of this model (see Figure 2.12). The mean-squared-error (MSE) loss (see Section 2.1.3) is used as the objective function. The regression objective function is deployed on the STS benchmark data that includes $8,628$ sentence pairs and is used to evaluate Semantic Textual Similarity (STS) tasks.

- **Triplet Objective Function:** Suppose an anchor sentence a, a positive sentence p, and a negative sentence n with the corresponding sentence embeddings s_a,s_p and s_n. The triplet loss tunes the network such that the Euclidean distance between s_a and s_p is smaller than the Euclidean distance between s_a and s_n:

$$max\left(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0\right)$$

ϵ denotes a margin that ensures that s_p is at least ϵ closer to s_a than s_n. The Authors set $\epsilon = 1$. This objective function is deployed on a Wikipedia dataset that comprises of *"weakly labeled sentence triplets: The anchor and the positive example come from the same section, while the negative examples comes from a different section of the same article"*. The triplet objective function with its structure is shown in Figure 2.13.

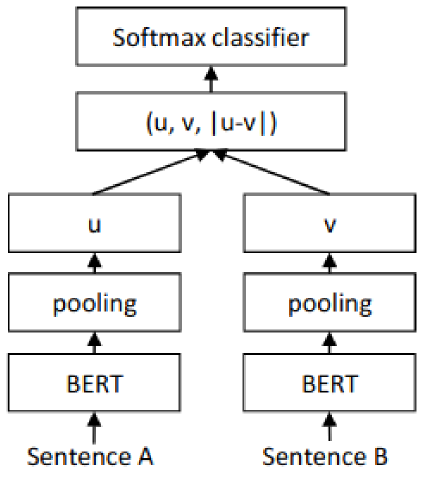


Figure 2.11:
SBERT architecture with the classification objective function that is based on a siamese network structure [19, Figure 1].

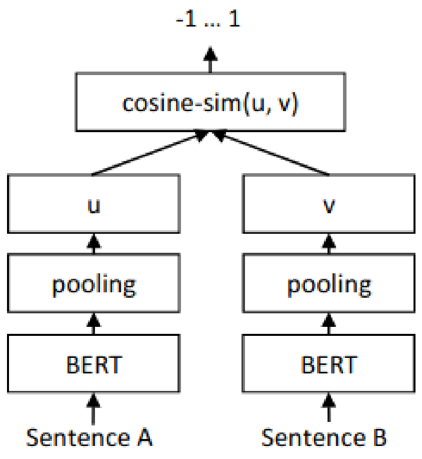


Figure 2.12:
SBERT architecture at inference, e.g., to compute similarity scores. This architecture is also used with the regression objective function [19, Figure 2].

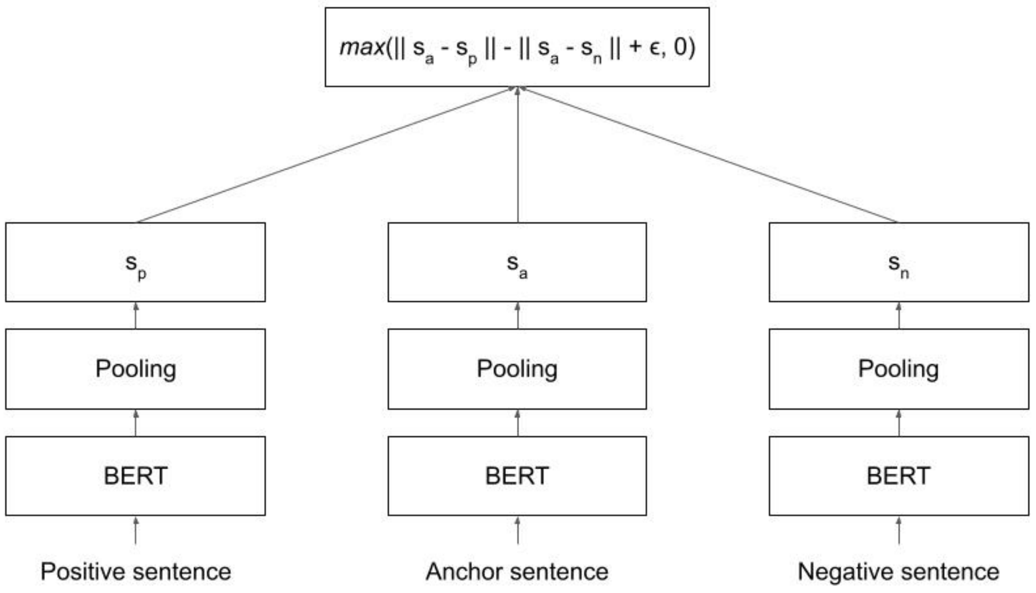


Figure 2.13: SBERT architecture with the triplet objective function that is based on a triplet network structure.

Siamese or triplet neural networks have the following properties [20][21]:

- Consist of two or three identical artificial neural networks that work in tandem to compute a similarity of two or three input vectors. Typically, the cosine similarity is computed.
- All networks are capable of learning the hidden representation of an input vector.
- They are feed-forward networks.

2.2 Information Retrieval

2.2.1 Retrieval Evaluation

The effectiveness of a retrieval process can be assessed by two key statistics:

$$Precision = \frac{\#\, relevant\; items\; retrieved}{\#\, retrieved\; items} = P\left(relevant \mid retrieved\right) \tag{2.1}$$

$$Recall = \frac{\#\, relevant\; items\; retrieved}{\#\, relevant\; items} = P\left(retrieved \mid relevant\right) \tag{2.2}$$

These notions can be made clear by examining the following contingency table [4, p.155]:

	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Then, $P = tp/\left(tp + fp\right)$ and $R = tp/\left(tp + fn\right)$.

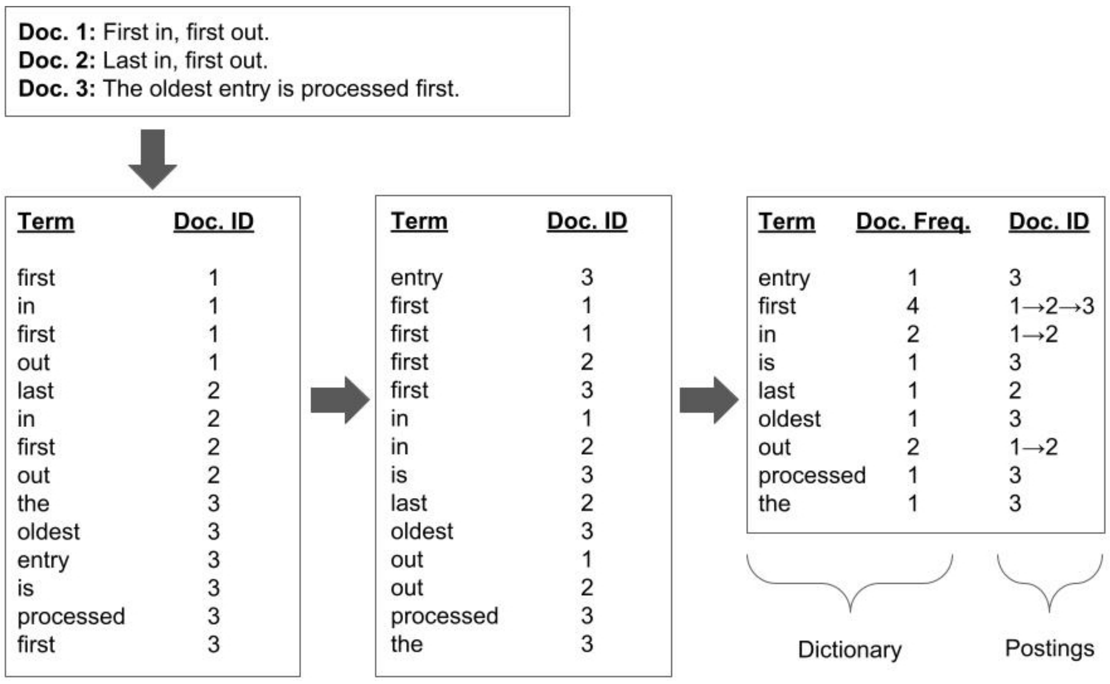
As an real life example, suppose the method retrieves 200 documents for a given query and out of these, 75 are actually relevant. For the given query there are 100 relevant documents in total. Thus, the exemplary retrieval has the precision $P = \frac{75}{200} = 0,375$ and the recall $R = \frac{75}{100} = 0.75$.

2.2.2 Inverted Index

"Usually, at the core of a classical search engine is a data structure called the inverted index, which closely resembles the physical index one can find in the back of a textbook" [22, p.22].

An inverted index is composed of two main pieces [22, p.22]:

- **Term dictionary:** A sorted list of terms that usually correspond to keywords (see Section 2.1.2), and occur across a collection of documents.
- **Postings:** Each term in the dictionary is associated with a *postings* list. This list contains an entry for each document in the collection that contains the corresponding dictionary term. An entry consists of the document ID and typically other information that is required by the system's scoring model, such as a term's frequency in the respective document [23].



Doc. 1: First in, first out.
Doc. 2: Last in, first out.
Doc. 3: The oldest entry is processed first.

Term	Doc. ID
first	1
in	1
first	1
out	1
last	2
in	2
first	2
out	2
the	3
oldest	3
entry	3
is	3
processed	3
first	3

Term	Doc. ID
entry	3
first	1
first	1
first	2
first	3
in	1
in	2
is	3
last	2
oldest	3
out	1
out	2
processed	3
the	3

Term	Doc. Freq.	Doc. ID
entry	1	3
first	4	1→2→3
in	2	1→2
is	1	3
last	1	2
oldest	1	3
out	2	1→2
processed	1	3
the	1	3

Figure 2.14: Building an inverted index by sorting and grouping.

The inverted index assumes text normalization (see Section 2.1), that is, converting a block of text into distinct, normalized tokens in order to build and query the data structure [24, p.197].

Figure 2.14 shows the process of building an inverted index [4, p.8]:

1. Initially, every document that is going to be indexed is converted into distinct, normalized tokens. These tokens are subsequently tagged by their document ID (left) and sorted alphabetically (middle).
2. Identical terms are then grouped, such that the resulting dictionary (right) does not contain duplicate terms, and such that the corresponding documents IDs for respective terms are combined to make up the postings. Each dictionary term has a pointer to its postings.

To sum up: An inverted index lists every unique word that appears in any document and identifies all corresponding documents it occurs in [25]. Since the term dictionary is sorted, one can efficiently find a term, and subsequently its occurrences in the postings structure. This is contrary to a forward index that lists terms related to a specific document.

2.2.3 Judgment List

A judgment list is an "expression of the ideal ordering" [26] and is essential for evaluating IR models. Furthermore, it is used to train supervised ranking models. For a given document collection, each entry in a judgment list corresponds to a judgment, that is, a query document ID, a relevant document ID and a corresponding relevance score. Based on the relevance scores, judgment lists can be categorized into two types:

- **Binary:** The binary values correspond to a document being *relevant* or *irrelevant*. Typically, binary judgment lists can be inferred from clickstream logs, questionnaires, etc. that require less human intervention. These automatic/semi-automatic generated judgment lists are cheap and abundant but noisy.

- **Scaled:** Usually, scaled judgments are human-rated and correspond to grades, e.g., 1 to 5, with 1 being *irrelevant* and 5 being *highly relevant*. Human-rated judgment lists are expensive and small but clean.

If judgment lists do not reflect the user's needs, the corresponding IR model is going to perform poorly. This goes after the *garbage-in-garbage-out* principle, thus judgment lists need to be of high quality.

2.2.4 Okapi BM25

Okapi BM25 is a term-weighting and document-scoring function that is commonly used in IR to rank a set of documents in relation to a given query [27].

$$BM25\left(Q, d_i\right) = \sum_{i=1}^{N} IDF\left(q_i\right) \cdot \frac{f\left(q_i, d_i\right) \cdot \left(k1 + 1\right)}{f\left(q_i, d_i\right) + k1 \cdot \left(1 - b + b \cdot \frac{doc\,length}{avg\,doc\,length}\right)} \tag{2.3}$$

In the following Equation 2.3 is broken down into its components which are then elaborated [28]:

1. Q denotes a query that consists of query terms: $Q = \{q_1, q_2, \ldots, q_n\}$, thus q_i refers to the i^{th} term in the query.
   d_i denotes a document in the document collection D: $d_i \in D$.

2. $IDF\left(q_i\right)$ is the inverse document frequency of the i^{th} query term that differs from the *IDF* formula in *TF-IDF* (see Section 2.1.2) and is defined as:

   $$IDF\left(q_i\right) = log\left(1 + \frac{(doc\,count - f(q_i) + 0.5)}{f(q_i) + 0.5}\right)$$

   The *doc count* refers to the number of documents in the collection D. $f\left(q_i\right)$ is the number of documents that contain the i^{th} query term.
   Essentially, the IDF component measures how often a term occurs at the collection level and penalizes common terms.

3. In the denominator $\frac{doc\,length}{avg\,doc\,length}$ represents how long a document is relative to the average document length. The document length is based on the number of document terms. The longer the document, the lower the corresponding BM25 score becomes.

4. Variable b in the denominator weights the effect of the aforementioned document length on the BM25 score.

5. $f(q_i, d_i)$ is the frequency of the i^{th} query term in document d_i. The higher $f(q_i, d_i)$, the higher the BM25 score becomes.

6. $k1$ is a variable that limits how much a single term can affect the score of a given document.

2.2.5 Ranking Evaluation

Common ranking quality measures are the *Discounted Cumulative Gain* (DCG) and the *Normalized Discounted Culumative Gain* (NDCG).
Suppose that T is the cutoff level for the returned results, such that only the first T results are considered. Furthermore, suppose that l_i is the relevance label for the i^{th} result, where $l_i \in 0, 1, \ldots, l$ and there exist a total order between labels $l > l - 1 > \ldots > 0$, such that the higher l, the higher the relevance. The DCG for a set of results that correspond to a given query is then defined as [29]:

$$DCG@T = \sum_{i=1}^{T} \frac{2^{l_i} - 1}{log(1+i)}$$

Suppose the maximum $DCG@T$ attainable for a query is denoted by $maxDCG@T$, then the $NDCG@T$ for a set of results, given the same query, is a normalized version of DCG, that is [29]:

$$NDCG@T = \frac{DCG@T}{maxDCG@T}$$

such that $NDCG@T \in \{0, 1\}$.

2.2.6 LambdaMART

LambdaMART is a ranking algorithm that can directly optimize rank specific cost functions such as $NDCG$ [30]. LambdaMART is a combination of *LambdaRank* and *MART*. For understanding, we are first going to review MART.

MART stands for *Multiple Additive Regression Trees*, also known as *Gradient Boosted Regression Trees*. This model is basically a weighted ensemble of *"weak learners"* that correspond to a regression tree [30]. Suppose we are given a data set $\{x_i, y_i\}$, $i = 1, \ldots, m$ where $x_i \in \mathbb{R}^d$ and the labels $y_i \in \mathbb{R}$. Then, the general objective is to find a ranking function $F(x)$ that minimizes the expected loss L for feature vectors x and labels y [30]:

$$F^*(x) = argmin_{F(x)} E\left[L\left(y, F(x)\right)|x\right]$$

The final model maps an input vector $x \in \mathbb{R}^d$ to a score $F(x) \in \mathbb{R}$ that corresponds to a weighted sum of N regression trees [29]:

$$F_N(x) = \sum_{i=1}^{N} \alpha_i f_i(x)$$

where each $f_i(x) \in R$ is a function modeled by a single regression tree and $\alpha_i \in R$ is the corresponding weight for the i^{th} regression tree. A given tree f_i maps a given x to a real value by passing x down the tree: the path (left of right) at a given node in the tree is determined by the value of a particular feature $x_{ij}, j = 1, \ldots, d$ and a threshold t, where $x_{ij} \leq t$ fall to the left leaf, and the rest fall to the right [29]. Each leaf is associated with a fixed value γ_{kn}, $k = 1, \ldots, L$, $n = 1, \ldots, N$, where L denotes the number of leaves and N the number of trees [29].
To minimize a loss function, MART performs gradient descent (see Section 2.1.3) in a stepwise manner: at each step we would like the model to change such that the loss decreases as much as possible. That "change" corresponds to the gradient of the loss function with respect to the current model m [30]:

$$g_{im} = \frac{\delta L[f(x_i), y_i]}{\delta f(x_i)}$$

The idea now is to build a regression tree f that models the gradient at a time step. This way, *"by adding such a tree to the ensemble, we effectively take a gradient step from the previous model"* [30]. To fit the tree, a squared loss is minimized [30]:

$$argmin_f \sum_{i=1}^{N} \left(-g_{im} - f(x_i)\right)^2$$

MART can be seen as a framework, as it allows for choosing the loss function and the corresponding gradient - this is where the *Lambda* part of LambdaMART comes into play [30].

Lambda The *Lambda* in LambdaMART refers to the loss function of the *LambdaRank* model [29]. For each training point pair i, j a value λ_{ij} is computed that acts as the gradient and can be thought of as a force that moves elements up and down a ranked list [30]. To summarize LambdaMART, the model uses MART as the Gradient Boosting framework with the λ values acting as gradients g. Furthermore, the paper specifies update rules for the leaf values γ_l that depend on the λ values of the training instances that fall in leaf l [30]. How the LambdaMART algorithm is computed in detail is shown in the LambdaMART paper by Burges [29] and is not further covered in this work.

2.2.7 Hierarchical Navigable Small World Graph

The approach presented by Malkov et al. [31] is developed for efficient approximate nearest neighbor search (ANNS) that is based on *Hierarchical Navigable Small World* (HNSW) graphs. In this graph structure, nodes represent data points, and links represent neighbor-connections with corresponding lengths/distances.

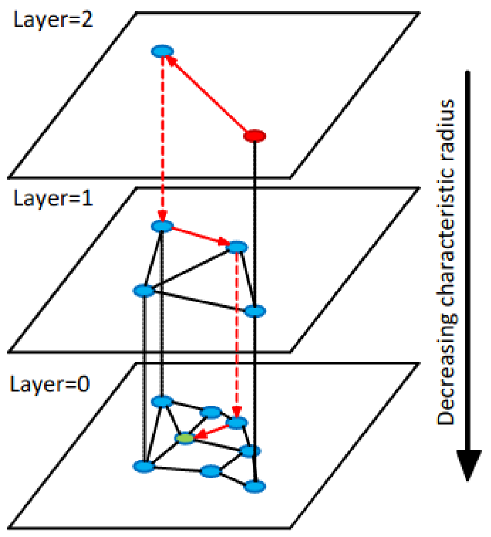


Figure 2.15: Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green). The Figure was taken from [31, Fig. 1.]

"The idea of Hierarchical NSW algorithm is to separate the links according to their length scale into different layers and then search in a multilayer graph" [31].

Figure 2.15 shows the greedy search process in a HNSW structure, starting from the top layer and finishing at layer 0 [31]:

1. The algorithm greedily traverses through the elements from the upper layer until a local minimum is reached.
2. Then, the search continues at the next lower layer, starting from the element that was the local minimum in the previous layer and the process repeats until the approximate nearest neighbor(s) is (are) found in layer 0.

The top layer has the longest links and in each subsequent layer the link lengths decrease (see Figure 2.15). At each layer only a needed, fixed portion of connections for each element is evaluated, which is independent of the network size, allowing logarithmic complexity scaling for the approximate nearest neighbor search [31]. To limit the scope of this work, further details about the search process and graph construction of this approach are omitted.

2.3 Related Work

In 1992 the *National Institute of Standards and Technology* (NIST) established the ***T**ext **RE**trieval **C**onference* (TREC) to solve two major problems in IR [32]:

1. IR researchers lacked large document collections with which to test IR systems and techniques.
2. There was a lack of standard evaluation methodologies stemming from a lack of coordination, which resulted in duplicative research and information asymmetries.

The TREC is formatted as a workshop that formulates specific information retrieval tasks, namely *Tracks*. Participants are provided *"a large test collection, uniform scoring procedures, and a forum for organizations interested in comparing their results"* [33].

The background linking task is part of the *TREC News Track*, which was established in 2018.

"The TREC News Track features modern search tasks in the news domain. In partnership with The Washington Post, we are developing test collections that support the search needs of news readers and news writers in the current news environment. It's our hope that the track will foster research that establishes a new sense for what "relevance" means for news search" [34].

Over the last three News Tracks starting from 2018, numerous teams participated in the background linking task, proposing different solutions to the problem.
In 2018, Naseri et al. deployed a probabilistic model that interpret a document's relevance as a probability, with respect to a query [35]. In 2019, Essam et al. addressed the problem by relying on graph-based analysis of the query text to extract representative keywords that are subsequently used to retrieve background links [36]. A year later, Boers et al. explored graph representations of documents based on named entities. This way, they could model newly introduced content in documents with respect to a query, which are more likely corresponding to background links [37]. In the same year, Deshmukh et al. combined a baseline *bag-of-words* model with a language model approach [38]. The authors used the baseline model to retrieve candidate documents. State-of-the-art sentence embeddings are then calculated for the candidates as well as the query. Finally, the candidates are ranked by their semantic similarity to the query, that is, the cosine similarity between the embeddings. Our work is similar to the aforementioned approach in that we also combine different retrieval methods and deploy state-of-the-art language models to make use of semantics. We think this is a promising direction for solving the background linking task. However, instead of just using semantic similarity for ranking, we deploy sentence embeddings as the unit of search as well, which we refer to as *semantic search*.

Surprisingly, the best performing models in each year, are based on rather simple and classic IR methods:

- 2018: The *htw saar* group's model is based on a classic ranked Boolean model with an inverted index. Keyphrases and named entities are extracted from the query document and are weighted by their occurrences in the document. These terms are subsequently used to query an inverted index [39].
- 2019: Fang et al. focused on using named entities for identifying background articles [40]. These entities are weighted based on surrounding words which they refer to as "context".
- 2020: Similar to 2018, Khloponin et al. used a ranked Boolean model [41]. Instead of keyphrases and named entities the model used the whole document text to query an underlying inverted index. Matching documents are ranked on corresponding *Okapi BM25* measures.

The stated IR methods find background links based on high lexical similarity, which Essam et al. showed to be not sufficient [3]. There is clearly an opportunity for improvements in the background linking task, as aforementioned classic IR approaches yield good results, but are certainly not optimal.

3 Background Linkage Modeling

In Section 3.1 we are going to reformulate the background linking problem from a more technical perspective and dissect it in its subproblems, which are then discussed in detail. Next, in Section 3.2 document models are defined that are going to be relevant throughout the chapter. In Section 3.3, we propose an IR framework in the form of a conceptual search pipeline that is divided into two levels. The subsequent Sections 3.4 and 3.5 are going to discuss these two levels respectively. Finally, the chapter is briefly summarized in Section 3.4.3.

3.1 Problem Setting and Objectives

The background linking task is defined by the *National Institute of Standards and Technology* (NIST) as the following:

Given a news story, retrieve other news articles that provide important context or background information. The goal is to help a reader understand or learn more about the story or main issues in the current article using the best possible sources [34].

Given a collection of articles and a query article, the background linking task can be modeled from a functional perspective:

$$\underbrace{(query\ article,\ collection\ of\ articles)}_{arguments} \underset{returns}{\rightarrow} \underbrace{ordered\ set\ of\ relevant\ articles}_{output}$$

To be mathematically precise, a collection of articles is denoted as $D = \{d_1, \ldots, d_n\}$, and the set of relevant result articles is denoted as $R = \{d_x, \ldots, d_y\}$, whereas the query article is denoted as d_i, such that $(d_i, D) \to R$, with the following properties:

- The query article d_i is not contained in the result set R: $d_i \notin R$.
- The query article d_i and the result set R are part of a collection of articles D: $d_i \in D$, $R \subseteq D$.
- The resulting articles R are ordered by relevance to the user's information need, in descending order, with the first document being the most relevant.
- The size of the output is expected to be much smaller (e.g., < 10) than the size of the input collection (e.g., > 10000): $|R| \ll |D|$.

For the background linking task it makes no sense to retrieve almost identical articles with respect to the query, but rather articles that share "some" common ground, and add new or more detailed information. Suppose the notion of similarity between query and documents could refer to lexical overlap, similar semantics and/or common meta-data. Now the problem is how do we define to what extent retrieved articles should be similar to a given query? What is the acceptance range for similarity? In other words, what is the upper and lower bound for similarity? To solve this problem, we make the assumption that the underlying document collection D does not contain (near-)duplicate news articles.
In practice a document collection could contain near-duplicates, for instance, after merging multiple collections from different news sources. Thus, this assumption may require automatic techniques for removing near-duplicates, because manual removal would be infeasible for large data. For example, in the *TREC 2019 News Track* a near-duplicate detection system was implemented, based on the *MinHash* and *locality-sensitive hashing* algorithms (automatic techniques for removing near-duplicate documents are not in the scope of this work). However, even with the use of near-duplicate removal techniques, in practice it cannot be guaranteed that all near-duplicates were removed. Certainly some near-duplicates are going to remain and some non-duplicates are going to be removed, as there is no perfect detection process.

For this reason, we relax the stated assumption and allow a small number of near-duplicates in practice that should not compromise the search performance too much.
With the aforementioned assumption, similarity can be treated as a proxy for relevance,. This way, we can focus on finding the most similar articles, according to specific similarity metrics for a given query, without worrying about (near-)duplicates - this solves the upper bound similarity. The lower bound can be ignored as we only consider the most similar documents.
Thus, we can simplify the definition of relevance in the context of the background linking task: the top-k similar articles, with respect to a query, are considered relevant background articles.

The background linking task is similar to an *ad hoc* IR task:

"In it, a system aims to provide documents from within the collection that are relevant to an arbitrary user information need, communicated to the system by means of a one-off, user-initiated query. An information need is the topic about which the user desires to know more, and is differentiated from a query, which is what the user conveys to the computer in an attempt to communicate the information need" [4].

However, the background linking task differs conceptually from the ad-hoc IR definition in two ways. First, there is no user actively trying to convey the information need in the form of a query. Instead, the query is derived from the input document automatically, using IR techniques, without direct user intervention. Secondly, the notion of *information need* is different. The information need for ad-hoc retrieval can be seen as *primary*, because it is defined as *"what the user wants"*. The information need in the context of background linking can be seen as *secondary*, which is *"what the user might find useful"*. Furthermore, it should be highlighted that the retrieval of relevant documents, given a query, is not sufficient. The retrieved documents have to be sorted by their relevance to the user's secondary information need in descending order, such that the most relevant documents are at the top.

To this end, we formulated the background linking task as a ranked retrieval problem, which we are going to refer from now on as a *search problem.*

How can this search problem be solved? For this complex task, we make use of the *divide-and-conquer paradigm.*

"A paradigm is a method of designing algorithms, a general approach to construct an efficient solution to a problem." [42].

This paradigm, divide-and-conquer, breaks a problem into subproblems that are similar to the original problem, recursively solves the subproblems, and finally combines the solutions to the subproblems to solve the original problem [43]. Following this paradigm, the search problem can be divided into two subproblems:

1. **Retrieval**
2. **Ranking**

This is comparable to the *Two-Level Retrieval Process* by [23]:

1. At the first level (retrieval), candidate documents are identified using an "approximate evaluation" taking into account only partial information on term occurrences and no query independent factors.
2. At the second level (ranking), promising candidates are "fully evaluated", allowing a more sophisticated and computationally expensive ranking procedure.

Wang et al. coined this process as *cascade ranking* and proposed their model without a fixed number of phases [44]. It is basically a tradeoff between effectiveness and efficiency:

"The model constructs a cascade of increasingly complex ranking functions that progressively prunes and refines the set of candidate documents to minimize retrieval latency and maximize result set quality" [44].

In other words, the search space is iteratively reduced by less expensive ranking functions, such that more expensive ranking processes can be run on fewer articles instead of, for instance, the whole collection. This is necessary in the times of Google, because users expect accurate search results in near real-time.

According to the divide-and-conquer paradigm, we divided the background linking task into two subproblems: retrieval and ranking. The main objective of this chapter is to "conquer" these subproblems, that is:

- For the retrieval subproblem: achieve high recall, in other words, retrieve as many relevant documents, with respect to a query, as possible.
- For the ranking subproblem: come as close to a user's ideal sense of ordering as possible, with strong emphasis on the first ranks. Furthermore, it is desired to improve precision at the ranking phase to reduce evaluation costs.

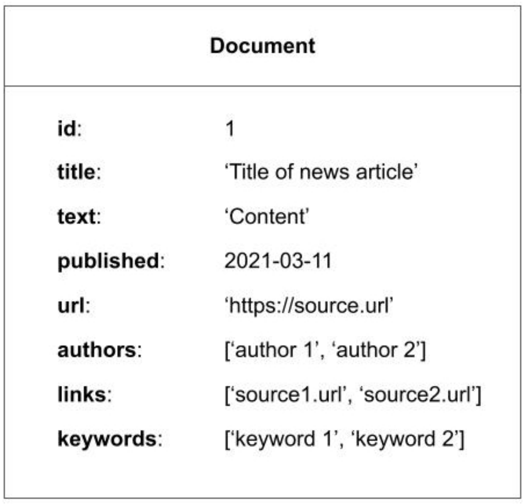
Additionally, this work has a strong emphasis on performance and scalability: proposed methods/solutions should be viable in practice.

3.2 Document Model

Before we can specify a background linking model, we need to define what is meant by "document". There can be various meanings, thus we are going to define several document models in this section.

"In search applications, the notion of a document is central, because documents are items being stored, searched, and returned" [22, p.18].

We define a *"general"* document model, according to Turnbull et al., as a set of named attributes, with corresponding values (see Figure 3.1) [22, p.18].



Document	
id:	1
title:	'Title of news article'
text:	'Content'
published:	2021-03-11
url:	'https://source.url'
authors:	['author 1', 'author 2']
links:	['source1.url', 'source2.url']
keywords:	['keyword 1', 'keyword 2']

Figure 3.1: General document model for news articles

The named attributes are described below:

- **id:** A unique identifier, which can be used to efficiently retrieve the respective document from the underlying index.
- **title:** Article's title that only contains natural language.
- **text:** Article's remaining textual content that contains only natural language.
- **published:** Article's time of publication.
- **url:** Article's URL-source.
- **authors:** Names of the article's authors
- **links:** Link references to other sources that are mentioned in the article.
- **keywords:** (Optional) keywords describing the content of the article. Some news outlets include them by default.

This document model resembles the form of a *Unified Modeling Language* (UML) class diagram, which can be regarded as structured data. The only difference is that there is no constraint on the data type for each attribute, as the shown values in Figure 3.1 are just exemplary. However, the attributes' values are differentiated to some extent. Manning et al. discriminate attributes into fields and zones: *"Whereas a field may take on a relatively small set of values, a zone can be thought of as an arbitrary, unbounded amount of text"* [4, p.110]. Possible field values are, for instance, the finite set of all dates or the authorship. The attributes *title*, *text*, *links* and *keywords* are considered as zones, whereas the remaining attributes are fields.

We refer to the defined document model as *general*, because it should be seen as a starting point from which other *specific* document models can be derived. This enables a flexibility to changing document model demands. Furthermore, its structured and thus homogeneous nature leads to a more streamlined processing. In addition, the included (meta-)fields, such as *published* and *authors*, could be useful in later processes.

In IR there is a plethora of retrieval methods, each having a specific document model. For instance, there are graph-based retrieval methods that describe each document as a graph, where nodes are usually named entities and edges represent co-occurrences between them [45]. However in this work, we are only going to cover two retrieval methods with the following specific document models:

- **Ranked Boolean Retrieval:** For the Boolean retrieval, the model views documents $d_i \in D$ as a set of terms: $d_i = \{t_1, t_2, \dots, t_n\}$ [4]. These terms are keywords that consist of one or more words. After index- and query-time, each retrieved document is represented as a vector $\vec{d_i}$. The vector representations allow computing scores according to a similarity measure (also known as vector space scoring) that are used for ranking. In this context each dimension corresponds to a specific term in the query document d_q. Suppose $d_q = \{t_1, t_2, \dots, t_m\}$, then $\vec{d_i} \in \mathbb{R}^m$.

The vector values in $\vec{d_i}$ represent the importance of a corresponding term and are dependent on the term's occurrence-statistics in the underlying document. We employ *Okapi BM25* for weighting each term (see Section 2.2.4). Possible values are:

- **Binary:** simply a 0 or a 1, whether the term occurs in the document or not. This representation is less suitable, as it ignores term frequencies.
- The **term frequency** in the document. This is also known as a bag-of-words model. Just looking at the term frequencies in a document is not sufficient, as this would treat every term the same. However, some terms are more important than others: for instance, stopwords occur more frequently than keywords, but are less significant to the content.
- A **similarity measure** based on the statistics of term occurrences at the document- and collection-level, such as *TF-IDF* (see Section 2.1.2) and *Okapi BM25*. These similarity metrics have been established in practice and allow terms to be weighted differently. *Okapi BM25* is preferred over *TF-IDF*, as it additionally incorporates document lengths.

• **Semantic Search:** Similar to a vector space model, each document is represented as a high-dimensional vector $\vec{e_i}$ in a common vector space (see Section 2.1.4). However, this specific model is not dependent on a (dynamic) vocabulary, as documents are projected to a *latent vector space* with a fixed dimension size h: $\vec{e_i} \in \mathbb{R}^h$ (see Section 2.1.4). Each document corresponds to a text embedding, based on the respective document's text.

At this point we defined a *general* document model (see Figure 3.1) and *specific* document models according to the aforementioned retrieval methods. These various document models are central to the *conceptual search pipeline* that is described in the following section.

3.3 Conceptual Search Pipeline

Based on the illustrated retrieval process by Broder et al. (see Section 3.1), we propose a *conceptual search pipeline* that is depicted in Figure 3.2. It can be seen as an IR framework, as we do not require which IR methods to use, but rather give a guideline which components are needed for search. This modularity allows for tailoring the search pipeline according to domain-specific needs. With this framework it is possible to deploy multiple IR methods in parallel and combine their results. This comes with the rationale that, in order to maximize recall it is beneficial to combine various retrieval methods, instead of relying on a single one. Though, this would require significantly more implementation efforts and hardware resources, such as memory, computation, network costs, etc. Furthermore, multiple IR methods introduce several features that could be beneficial for the ranking task. While the recall can only increase with the union of retrieval sets, the precision and ranking measure (that measures the distance between the actual and the expected ordering of documents) could become worse. Thus, it remains an open question which maximum combined retrieval size to choose if recall is prioritized.

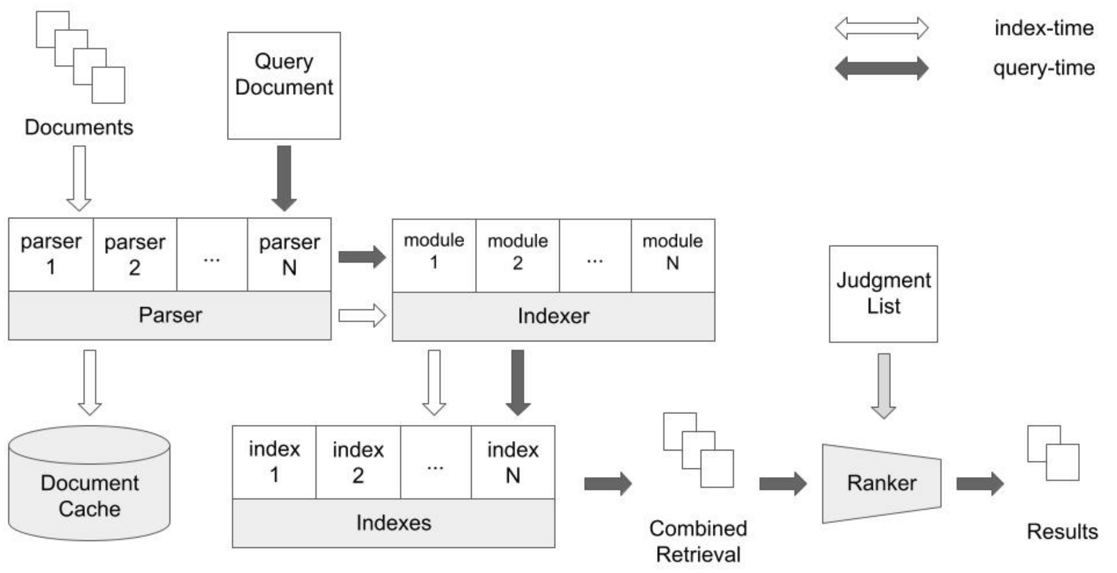


Figure 3.2: Conceptual search pipeline

Our conceptual considerations also include some technical aspects. The depicted components in Figure 3.2 are described at a high-level in the following:

- **Document Collection:** This can be any large collection of structured, semi-structured or unstructured (textual) data.

- **Parser:** As major news consumption moved online, digital news articles can contain videos, audio recordings, images etc. These audio- and visual data are ignored and only textual data are considered in this work. Depending on the information source, articles could have different text formats:

  – Relational database: A news article has a formal structure associated with an underlying data model

  – In the browser: Typically unstructured (textual) data in the form of an HTML-file.

  – Other formats: XML and JSON, etc.

  The Parser-component is responsible for analyzing the documents' textual data to determine the format, and subsequently build a data structure based on the *general* document model (see Section 3.2). This way, different text formats can be supported. The component is made up of Parser-modules that can be easily extended and replaced.

- **Indexer:** Each retrieval method requires a *specific* document model (see Section 3.2) before querying or indexing. The Indexer-component is responsible for deriving specific document models from the respective *general* document model for each document. Indexer-modules are dependent on the deployed indexes.

- **Indexes:** Are responsible for storing documents in a way such that the retrieval is efficient. The indexes respond to queries with a combined retrieval set of relevant documents. Depending on Indexer-modules, indexes can be extended or replaced.

- **Ranker:** Ranks the combined retrieval, according to their relevance to the user's information need. Finally, the ordered list is pruned to return the top-k relevant documents. The underlying ranking model is trained and optimized on a scaled *Judgment List* (see Section 2.2.3). An appropriate ranking model needs to be deployed.

- **Document Cache:** Stores documents, according to the *general* document model and allows efficient retrieval by the respective document's unique identifier. This component is necessary for enabling other components to request further or missing information about a document, as components might only retain partial information to avoid memory overhead.

Initially, each document needs to pass through the Parser- and Indexer-component. For each document, the Parser-component parses and converts it to the general document model (see Figure 3.1). Based on that, Indexer-modules in the Indexer-component build bespoke specific document models, that are needed for indexing or querying the corresponding index in the Indexes-component. Both components are inter-dependent: an indexer-module m_i corresponds to an index i_i, as a module-index pair refers to a distinct retrieval method.

3.4 First Level: Retrieval

For the background linking task, we propose an IR framework that is not restrictive of which retrieval method to use. To showcase this framework, we choose to deploy two different retrieval methods. In this section, we are going to discuss both retrieval methods in detail and explore how they fit in the bespoke framework.

At this stage in the pipeline, documents are already indexed in respective indexes. We are going to focus on the search pipeline, starting at query-time.
The retrieval phase starts with the query document as an input. After going through the Parser- and Indexer-component, the query is converted into two specific document models: a term-based vector and an embedding (see Section 3.1), corresponding to the two aforementioned retrieval methods, namely ranked Boolean model and semantic search respectively.

Both query representations are then used to query respective indexes, that is, an inverted index (see Section 2.2.2) for the ranked Boolean model and a vector index (see Section 2.2.7) for the semantic search. Following the proposed framework, the resulting retrieval sets are combined and make up the input for the second level: Ranking (see Section 3.5)

It can be observed in the Figure 3.2 that the final search results are based on the combined retrieval. If the retrieval has low recall, even the best pruning and ranking methods would be useless, as the search engine would perform poorly nevertheless. In fact, the whole search process is a chain of processes, each depending on the process before. If there is bad performance at one point in the chain, it is going to propagate further down, eventually to the results. This tight coupling of dependencies makes search problems challenging. The objective at this level is to achieve a strong baseline retrieval, prioritizing high recall. However, the size of the retrieval sets should still be orders of magnitudes less than the size of the whole collection.

In the context of IR there is a rich choice of information, approaches and techniques. This raises the challenge of how to use them together and fully explore their (combined) potential. Information retrieval models can be categorized in three categories, according to their mathematical basis [46]:

- **Set-theoretic Models:** Documents are represented by sets that contain terms. Similarities are derived using set-theoretic operations.
- **Algebraic Models:** Documents are represented as vectors, matrices or tuples. These are transformed using algebraic operations to a one-dimensional similarity measure.
- **Probabilistic Models:** Document's relevance is interpreted as a probability. Document and query similarities are computed as probabilities for a given query.

At the conceptual level it is not possible to predict which retrieval models perform best for the background linking task. To limit the scope of this work, we do not cover probabilistic retrieval models and choose the two aforementioned retrieval models that are presented and discussed in the following sections.

3.4.1 Ranked Boolean Retrieval

The classic Boolean retrieval model can be categorized as a *set-theoretic* model. Terms are combined with the Boolean operators *AND*, *OR*, and *NOT*, to form a query.

In the context of the pipeline, we are at query-time. At this point, the query document is in the form of the *general* document model (see Section 3.2). However, the Boolean retrieval model views each document as a set of terms (see Section 3.2). This requires the query document with blocks of text to be converted into distinct and normalized tokens (see Section 2.1). Usually, these tokens are keywords, thus this common retrieval method is also known as *keyword matching*: Keywords that are extracted from the query document, are used as a query to find documents in the collection with matching terms.
Suppose d_q denotes a query document in the form of the general document model, then it needs to be converted to $d_q = \{t_1, t_2, \ldots, t_m\}$ with t_i denoting a distinct keyword. The mapping of general document models to the specific ones is the Indexer-component's responsibility.

There are many methods for keyword extraction available. We choose to extract keywords based on their *TF-IDF* score, as it incorporates the whole document collection in contrast to the *RAKE*-algorithm (see Section 2.1.2). Furthermore, keyword extraction based on *TF-IDF* needs less intensive computation and implementation efforts than a trained classifier.
Some news articles already contain keywords or tags that are placed by the author to sum up the content. These human-annotated keywords could eliminate the need for a keyword extraction, under the premise that human-annotated keywords perform better than algorithmically extracted ones.

This needs to be tested in the experimental evaluation Chapter 4.

As we prioritize high recall, a simple Boolean query that only uses the *OR*-operator is sufficient. This specific Boolean query is described in Algorithm 1: Initially, all query terms and their corresponding postings are identified in the dictionary of the underlying inverted index (see Section 2.2.2). The subsequent union of the postings make up the resulting list of matching documents.

```
boolean_match_or $(I, Q)$
Input: Inverted Index $I$, Query terms $Q$
Result: Matching Document IDs $P$
$P \leftarrow \emptyset$;
$T \leftarrow get\_dictionary\,(I)$        // list of index terms in dictionary
for $q \in Q$ do
    if $term\_found\,(T, q)$ then
        $P_t \leftarrow get\_postings\,(I, q)$// dynamic list of matching document
            IDs
        for $p_t \in P_t$ do
            if $p_t \notin P$ then
                $P \leftarrow P \cup \{p_t\}$;
            end
        end
    end
end
return $P$;
```

Algorithm 1: Function describing a Boolean query based on the OR operator, run on an inverted index.

The main drawback of the Boolean retrieval model is that it often *"retrieves either too many or too few documents due to sensitive nature of the boolean logic that responds rigidly to the absence or presence of a single term"* [47].
The problem of *"too few"* results becomes apparent when the index terms in the dictionary of the inverted index are not preprocessed properly, that is for instance, lowercasing and stemming/lemmatization:

- *"Apple"* and *"apple"* would not match, even though they only differ in the casing
- *"jumps"* and *"jumped"* would not match, even though they share the same root word *"jump"*

These NLP normalization techniques (see Section 2.1) are important for relaxing matching rules and thus mitigating against the sensitivity issues of the Boolean retrieval.
The problem of *"too many"* results could be, for instance, due to the use of the *OR*-operator, typically in combination with many query terms. The Boolean retrieval only returns matching documents based on a query and does not differentiate how well each document matches the query. For example, a document that matches only 1 term in a 100-term query, is in the same result set as a document that matches all the 100 terms. It is apparent that the result set for long queries becomes very large. We want to know how relevant each document is, with respect to the query, and get the most relevant results. This is where the vector space model comes into play, which can be categorized as an *algebraic model.*

After query-time, we now have a retrieval set of matching documents with respect to a Boolean query. According to the vector space model, each retrieved document needs to be represented as a vector $\vec{d_i}$ in a common vector space. These vector representations are used for ranking (see Section 3.2). In this context each dimension corresponds to a specific term in the query document d_q. Suppose $d_q = \{t_1, t_2, \ldots, t_m\}$, then $\vec{d_i} \in \mathbb{R}^m$. The vector values v_i in $\vec{d_i}$ represent the importance of a corresponding term and are dependent on a term's occurrence-statistics in the underlying document. We deploy *Okapi BM25* for weighting each term. The ranking score corresponds to the sum of a vector's elements:

$$Score\left(\vec{d_i}\right) = \sum_{i=1}^{m} v_i \tag{3.1}$$

We extend the classic Boolean retrieval: After the retrieval of the matching documents, a simple ranking and pruning step is introduced (see Algorithm 2), thus we refer to this retrieval as **Ranked Boolean retrieval**.

```
boolean_retrieval_ranked $(I, Q, k)$
Input: Inverted Index $I$, Query terms $Q$, Maximum retrieval size $k$
Result: Matching Document IDs $P$
$P \leftarrow$ boolean_match_or $(I, Q)$                    // see Algorithm 1
$P \leftarrow$ sort_descending $(P)$ // assuming BM25 score for each element
$P \leftarrow \{P[0], \dots, P[k]\}$;
return $P$;
```

Algorithm 2: Boolean query with additional ranking and pruning.

The ranking is done by computing a ranking score for each document vector according to Equation 3.1. The documents are then sorted by their score in descending order and subsequently pruned to get the top-k results that are going to be part of the Combined Retrieval (see Figure 3.2).

The ranked Boolean retrieval method is going to serve us as a baseline, mainly because of its simplicity and reasonable performance, as it is established in IR practice. We are going to improve upon the baseline and evaluate how the semantic search model compares against it at conceptual level and later in the experimental phase.

3.4.2 Semantic Search

This section is going to discuss an alternative retrieval method to the classic Boolean retrieval: **semantic search**. Initially, we are going to describe drawbacks of the Boolean retrieval and the motivation behind using semantics in search. Furthermore, we are going to examine various semantic search approaches and subsequently focus on a sentence embedding based approach. Then a vector index is presented, which is necessary for efficient retrieval of high-dimensional vectors. Finally, we are going to show how semantic search fits in the search pipeline.

Drawbacks of Classical Retrieval The ranked Boolean retrieval has the drawback of ignoring the grammatical order of the respective terms.

In natural language the order of terms can be crucial for understanding, and ignoring them can lead to ambiguity or even alter the intended meaning.
Furthermore, the model cannot handle synonymy and polysemy [4, p.413]:

- **Synonymy** refers to different words sharing the same meaning. As the Boolean model treats different words as separate entities (or separate dimensions), it fails to capture the relationship between synonyms, and subsequently underestimate the true similarity that a user would perceive.
- **Polysemy** refers to an ambiguous word having multiple meanings. As the meaning for the same word could change throughout the document, both models would overestimate the similarity that a user would perceive.

Additionally, the Boolean retrieval method is based on the heuristic that lexically similar documents are relevant. A qualitative analysis of news articles' relevance in the context of background linking by Essam et al. shows that lexical overlap as an heuristic is not sufficient on its own for assessing relevance [3]. In their study, they could find many background articles that shared high lexical similarity with the query, but were less relevant or even irrelevant to the reader. Vice versa, they found many background articles with low lexical similarity, given a query, but had crucial relevance. Nevertheless, they could show a positive, but less strong, correlation between lexical similarity and the relevance levels of the background articles, and suggest that lexical similarity could be used as a relevance signal.
The stated issues are mainly due to the term-centric nature of the previous approach. We are going to address them by introducing another retrieval model: *semantic search.*
In contrast to the Boolean retrieval model that looks for literal matches of query terms in the documents, semantic search is about search with *"meaning"*. That is, *"understanding the query, instead of simply finding literal matches, or representing knowledge in a way suitable for meaningful retrieval"* [48].
Semantic search might handle synonymy and polysemy much better, as it 'understands' the meaning of terms and could make use of their underlying context.

Semantic Search Approaches The most common ways of implementation are:

- **Latent Semantic Indexing**(LSI): Also known as Latent Semantic Analysis (LSA), views document collection D as a term-document matrix M of size $|D| \times v$, with v being the size of distinct terms in the collection (vocabulary size). Values in the matrix refer to a weight of a term in the respective document, and usually depends on its occurrence statistics (e.g., TF-IDF or BM25). The idea is to use co-occurrences of terms found in the matrix, to capture their latent semantic associations. For the indexing step, a method called *Singular Value Decomposition* (SVD) is deployed to construct a low-rank approximation M_k, for $k << v$.
  "When forced to squeeze the terms/documents down to a k-dimensional space, the SVD should bring together terms with similar co-occurrences"
  [4, p.415].
  Queries and new documents are mapped to their representation in the reduced latent space. As query and documents are vectors, they can be compared using cosine similarity.
  The main drawbacks of this approach are the computational costs of performing SVD on a large matrix and the index' inflexibility to change. After the low-rank approximation, the index M_k fails to capture the co-occurrences of newly added documents, as well as newly introduced terms. This is because new documents are simply mapped to the reduced latent space, otherwise the whole indexing process has to be repeated for each new document [4, p.414]. In other words, the low-rank approximation M_k can be seen like an approximate snapshot of the term-document matrix M, and thus has a static nature. This is impractical in real-world usage, as the vocabulary size is usually subject to change. Thus, this approach is not implemented in this work.

- **Knowledge-graph-based** approaches: A knowledge graph is a representation of semantic relations between entities that can correspond to persons, organizations, events, etc. The nodes correspond to the aforementioned entities and the edges represent semantic relations between them.

Documents and queries are usually represented as sets of such entities and their semantic similarity can be derived from the knowledge graph. Knowledge graphs store rich semantics in semi-structured formats and have great potential in improving text understanding and search accuracy. This approach requires access to huge knowledge graphs and is not covered in this work.

- Based on **text embeddings**: Documents and queries are initially represented as either unstructured free text (sentences) or lists of words. Machine learning methods are then used to calculate low-dimensional representations for each sentence or word, namely sentence- and word embeddings (see Section 2.1.7). Thus, both query and documents are vectors and can be compared using a distant metric. In this context, the *cosine similarity* is the appropriate distance metric, because it compensates for the effect of different document lengths (see Section 2.1.7). The concept of text embeddings is similar to the *LSI* method, in which data points are projected to a multi-dimensional *latent vector space* (see Section 2.1.4). In contrast to LSI, embeddings do not suffer from newly introduced documents or changing vocabulary, as their fixed dimension size is independent from the vocabulary size.
  Sentence embeddings are preferred over word embeddings in this context, as word embeddings are still term-based and thus ignore the grammatical order. With its promising properties we are going to deploy **semantic search based on sentence embeddings**.

Characteristic Text for Sentence Embeddings For the keyword matching retrieval, characteristic terms were identified that roughly represent the underlying document. The same has to be done for sentence embeddings: There is a need to identify characteristic text in sentence length that represents the underlying document. As embeddings do not work well on large text sizes, it is not advisable to embed the whole textual content. On the one hand, this would be computationally expensive and on the other hand, sentence embeddings are limited by a certain text size that roughly corresponds to the size of a sentence.

There is a lot of freedom to choose characteristic text to calculate embeddings on at both index- and query time. In fact, for n components there are n^2 possible combinations.
Typically, news articles start with a summary of the main story, followed by detailed coverage of major aspects of the story that are structured into paragraphs [49]. Thus, sensible text components are:

- title
- first paragraph
- section titles
- human-annotated keywords
- algorithmically extracted keywords

The experiments in Chapter 4 are going to reveal which text components and combinations are most suitable at index- and query time.

Alternative Index Structure As we are going to deploy sentence embeddings, there is a need for an alternative index that can efficiently store and retrieve, high-dimensional vectors.
Classic IR methods, such as the Boolean retrieval (see Section 3.4.1), assume an inverted index (see Section 2.2.2) as the underlying retrieval structure. The inverted index is optimized for efficient retrieval of exact terms, but is less effective for other retrieval tasks that require ranking. We are especially interested in finding top-k elements from a dataset with minimal distance to a given query - this is known as *K-Nearest Neighbor Search* (K-NNS)[31]. As exact solutions for K-NNS are limited due to the "curse of dimensionality", the concept of *Approximate Nearest Neighbor Search* (K-ANNS) was proposed, which allows a small number of errors [31]. Inverted indexes usually have to reduce the search space, such that more elaborate comparisons can be made. This is essentially a tradeoff between effectiveness and efficiency. A number of heuristics can be used to reduce the search space, without harming the recall too much, e.g., only considering documents that match at least two query terms. Nevertheless, potential candidates are going to be discarded inevitably. Obviously, one could evaluate all documents in the collection.

That would be more accurate, but scales linearly with the collection size, instead of a constant. It is clear that an alternative index structure is needed. We formulate the following index requirements:

- Perform K-ANNS efficiently (better than linear complexity), without filtering out potential candidates as an heuristic.
- As we represent documents as embeddings, the new index structure should support high-dimensional vectors.
- Index updating should be supported, e.g., insertion (required) and deletion of elements (optional)

The *Hierarchical Navigable Small World* (HNSW) structure, (see Section 2.2.7) proposed by Malkov et al. [31], satisfies all our requirements for the semantic index. HNSW is based on graph structures and has huge potential in the context of information retrieval. First and foremost, it is extremely efficient while maintaining high recall: the search complexity scales as $O\left(log\left(N\right)\right)$, whereas the index construction time scales as $O\left(N \cdot log\left(N\right)\right)$. Furthermore, it supports incremental index construction and element deletions. Additionally, it can handle arbitrary many data points, as well as high-dimensional vectors. There is no additional ranking step necessary, as the retrieval is already ranked. The difference in performance can be explained by the fact that HNSW evaluates documents at index-time and indexes similar documents "closer" to each other, in contrast to inverted indexes that evaluate documents at query-time. As there are no conceivable drawbacks, HNSW could make traditional index structures, such as the inverted index, redundant in the context of ranked retrieval.
Generally speaking, a retrieval method can be dissected into 3 parts: a document model, a similarity metric (or definition of relevance), and an index structure for efficient retrieval. As the HNSW index structure is close to optimal, it only boils down to the document model and similarity metric.

We implement a retrieval method, namely semantic search, that is based on sentence embeddings with the following promising properties:

- vocabulary-independent
- invariant to synonymy
- might handle polysemy better due to context understanding
- incorporate grammatical order, in contrast to term-based approaches

Conceptual Search Pipeline Similar to the Boolean retrieval, we are now at query-time. At this point, the input is a query document in the form of the *general* document model (defined in Section 3.2). The Indexer-component receives that input and is responsible for mapping the general document models to the specific ones. The Indexer-component's output is a sentence embedding for each document, as required by the semantic search model (see Section 3.2). For this, the sentence embedding is calculated, based on extracted characteristic text in the query document. Suppose d_q denotes a query document in the form of the general document model. Then, it needs to be converted to a sentence embedding $\vec{e_q}$ in latent vector space with a fixed dimension h: $\vec{e_q} \in \mathbb{R}^h$.

Even with the stated beneficial properties, it is not clear whether semantic search performs better than the ranked Boolean model (in terms of retrieval recall). For example, even though the Boolean model is based on lexical overlap instead of semantics, the use of multiple keywords could roughly correspond to the correct context. To come to a conclusion, which approach performs best, they have to be experimentally evaluated (see Chapter 4). Although it is interesting to see which retrieval approach performs best, it is not practical to discard worse performing methods. The contrary, it makes sense to combine different retrieval methods, to achieve higher recall. Suppose r_t denotes the retrieval set for a specific index $i \in I$, where I denotes the set of indexes in the search system. Then, the combined retrieval set can be formulated as $r_c := \bigcup_{i \in I}\{r_t \in i\}$.

3.4.3 Summary

In this Section 3.4 we covered two drastically different IR methods, namely ranked Boolean retrieval and semantic search. The former retrieves documents based on lexical overlap and an inverted index as the underlying index structure. The latter relies on documents that are represented as sentence embeddings to capture their meaning, and a vector index in the form of HNSW-graphs (see Section 2.2.7). Together, they make up the retrieval process, that is, the first level of the pipeline. Their result sets are combined, resulting in a Combined Retrieval (see Figure 3.2), which is the input for the next phase: *ranking*.

3.5 Second Level: Ranking

The objective in this second phase is to sort results from the first-level retrieval, such that the ranking comes as close as possible to the user's ideal sense of ordering for the documents associated with the query [26].
Following the conceptual search pipeline, the first level retrieval process outputs a combined retrieval set that serves as an input for the second level ranking step. In the second level, the ranker-component receives the combined retrieval set, together with a judgment list, as inputs. The documents in the combined retrieval set are subsequently ordered by the trained ranking model, based on extracted document features. Finally, the ordered set is pruned and the top-k result are returned.

3.5.1 Motivation

The second level ranking step is redundant if the covered retrieval methods are deployed exclusively:

- The ranked Boolean retrieval method already ranks its elements regarding a ranking score (see Equation 3.1).
- The semantic search retrieval, with the underlying HNSW-index, returns approximate nearest neighbors that inherently have an order.

However, with several approaches we want to combine each retrieval set by their union. As each retrieval set might have their own specific ordering, it becomes unclear how the combined result set should be sorted. The second level ranking step is necessary for the framework, to support combining multiple IR methods.

3.5.2 Features

Features can be seen as the *raw material of relevance* [26], which can be incorporated for ranking. In the context of ranking and IR there are three types of features:

- **Query-independent** features or **static** features describe the underlying document and can be computed beforehand, during index-time, e.g., published date, document length, etc.
- **Query** features describe the query and need to be computed at query-time, e.g., query length.
- **Query-dependent** features describe the relationship between both the underlying document and the query, e.g., cosine similarity, Okapi BM25 score, etc.

Coming up with features and selecting effective ones are disciplines of their own, also known as *feature engineering* and *feature selection* respectively. If the features are poor, even good judgments are not going to help predicting patterns in the relevance scores, thus resulting in a bad search experience. This goes, again, after the *garbage-in-garbage-out* principle [26]. Nowadays, modern systems use a great number of features. In 2008 a former vice president of Google was cited by the New York times, saying that Google was using over 200 of such features [50]. In the context of this work, the number of features are limited to textual features that are derived from the underlying document, because clickstream logs, or user-based data in general, are not feasible.
Depending on the deployed retrieval methods, each approach might introduce a different relevance signal. Thus, results in the retrieval set might contain different (many) features.

We covered the ranked Boolean model and semantic search retrieval, both introducing an Okapi BM25 score and a cosine similarity score respectively. The BM25 score is already computed at this stage for documents retrieved by the Boolean retrieval model. Similarly, the cosine similarity score is already computed for documents retrieved by the semantic search retrieval. Thus, at most one of both features needs to be computed for a document in the combined retrieval set, which is beneficial for the performance.

As the ranked Boolean model is seen as a baseline retrieval method in this work, the corresponding BM25 score is seen as a baseline feature. It is possible to deploy multiple features for the ranking task at once. However, this makes analyzing the contribution of each feature to the ranking unclear. Instead, we use BM25 as a baseline feature, and iteratively trying to improve ranking accuracy by adding features one by one. Like a greedy algorithm, a feature remains if the ranking accuracy improves. In the context of the two covered retrieval methods, we are going to evaluate the ranking accuracy of the BM25 feature (corresponding to the ranked Boolean model) exclusively, followed by adding the cosine similarity feature (corresponding to the semantic search model) to evaluate the ranking accuracy of both features. Chapter 4 is going to reveal to which extent semantic search, with the corresponding cosine similarity feature, is going to improve the ranking.

We do not impose the use of certain features in our framework, though it is highly recommended to only use static features at this stage, as they are already computed. Thus, static features are beneficial for the performance, such that each element in the combined retrieval could be evaluated for ranking, without the need for cascade ranking. This way, the two level approach should be sufficient.

We use two features for ranking: BM25 and cosine similarity. Each document in the Combined Retrieval (see Figure 3.2) needs to be represented as a feature vector, that is, a vector containing the aforementioned features. This way, the feature vectors from the Combined Retrieval can be used for ranking (similar to the vector space scoring in the ranked Boolean model).
As both features are only partially calculated at this point, missing features need

to be calculated for each document. The Ranker-component is responsible for extracting features for ranking. After extracting features for each document in the Combined Retrieval we now have a set of feature vectors that can be ranked by a ranking model that is introduced in the next section.

3.5.3 Learning to Rank

At this point in time, we have a set of feature vectors that we want to order. Suppose we have an error function that calculates how far a ranking is from an ideal ordering. Now the question is how do we derive a ranking score based on selected features, on which documents can be compared and ordered, such that the error function is minimized? As an example, the ranked Boolean model's ranking score corresponds to the sum of a vector's elements (see Section 3.4.1). However, this ranking score does not differentiate features by weighting them, and does not guarantee to minimize the error function. Furthermore, it is not effective to tune feature weights manually. Instead, we employ a machine learning technique that learns how to rank the retrieved documents automatically and in a way that minimizes an error function - this is also known as *Learning to rank* (LTR).

Li formally describes the ranking task as the following [51]:

- Suppose $\{q_1, q_2, \ldots, q_m\}$ denotes a set of queries for the training, where q_i is the i-th query, $D_i := \{d_{i,1}, d_{i,2}, \ldots, d_{i,n}\}$ denotes the set of documents associated with q_i. Furthermore, suppose D_i is identified by integers $\{1, 2, \ldots, n\}$.
- Suppose a permutation or ranking list p_i on D_i is defined as a bijection from $\{1, 2, \ldots, n\}$ to itself. P_i denotes the set of all possible permutations on D_i.
- Ranking is basically selecting a permutation $p_i \in P_i$ for a given query and associated documents D_i by using scores given by a ranking model $f(q_i, d_i)$.

The objective is to train a ranking model that can assign a score to a given query-document pair, such that a loss function (see Section 2.2.5) is minimized.

We are going to deploy *LambdaMART* (see Section 2.2.6) as the underlying ranking method, which is a boosted tree approach and has proven to be successful in solving real world ranking problems [29].

Training, Validation & Test Set LTR can be categorized as supervised machine learning (see Section 2.1.3) and thus has a training and testing phase, with corresponding training and test data. We use the formal description of the training data by Li [51]:

- The training/test data consists of queries, documents and relevance ratings, where each query is associated with a number of documents, and each relevance rating is associated with a document, with respect to the query
- The relevance judgments or grades are denoted by labels $y = \{0, 1, 2, \ldots, l\}$. There exists a total order between grades $l > l - 1 > \ldots > 0$ [51]. The scaled relevance judgments are usually human-rated. Generally speaking, the higher the grade, the more relevant is the respective document to a given query.
- Suppose $y_i := \{y_{i,1}, y_{i,2}, \ldots, y_{i,n}\}$ denotes the set of labels associated with q_i and n denoting the size of y_i and D_i [51]. The training set can then be formulated as $S := \{(x_i, y_i)\}_{i=1}^{m}$, where $x_i := \{x_{i,1}, x_{i,2}, \ldots, x_{i_n}\}$ denotes a set of feature vectors [51].
- A feature vector $x_{i,j} := \theta\,(q_i, d_{i,j})$ is created from each query document pair $(q_i, d_{i,j})$, $i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$, where θ denotes the feature functions. That is to say, features are defined as functions of a query document pair.

To avoid overfitting of the ranking model, training data S needs to be split into three sets, that is, training, validation and test set: $S = S_{training} \cup S_{validation} \cup S_{test}$.

To this end we selected two features, namely the BM25 score and cosine similarity score for a document, with respect to a query. Based on these features the Ranker-component derives a set of feature vectors from its inputs: the Judgment List and the Combined Retrieval (see Figure 3.2). The feature vectors from the

Judgment List are used to generate training data for the ranking model.
The ranking model needs to be already trained before the ranking process can start. After training, the ranking model ranks the set of feature vectors from the Combined Retrieval. Finally, the Ranker-component outputs a ranked and (optionally) pruned retrieval set.

3.5.4 Evaluation

In general, the performance of ranking models is evaluated by comparing the actual ranked output of the model and the optimal ranking list given by the test set. The evaluation measures *DCG* and *nDCG* are widely used in IR (see Section 2.2.5). According to the TREC 2020 guidelines [52], the primary metric for evaluating the ranking model (in the context of the background linking task) is $nDCG@5$.

3.6 Summary

In this chapter we proposed a general framework, in the form of a conceptual search pipeline, for solving the background linking task. The pipeline is differentiated at index- and query-time:

- **At index-time:** Documents of a collection go through the Parser- and Indexer-component to be converted into specific document models that are subsequently indexed in corresponding indexes that make up the Indexes-component (see Algorithm 3).
- **At query-time:** Similar to the index-time, a query document initially goes through the Parser- and Indexer-component to be converted into specific document models that are used to query corresponding indexes in the Indexes-component. The union of the retrieval sets of each index in the Indexes-component is passed to the Ranker-component as its input. The Ranker-component then extracts features and subsequently converts the input into a set of feature vectors. These feature vectors are afterwards ranked by a ranking model that was trained by the provided Judgment List. Finally, the Ranker-component outputs the (pruned) ranked results (see Algorithm 4).

```
1  $index\_documents\,(D)$
   Input: Collection of Documents $D$, Indexes $I$, Document Cache $C$
   Result: Updated $I$, updated $C$
2  for $d \in D$ do
3      $f \leftarrow parse\,(d, P)$           // parse document format with Parser
4      $d_g \leftarrow map\_general\,(d, f)$     // map document to general document
           model
5      update $C$ with $d_g$;
6      for $i \in I$ do
7          $d_s \leftarrow map\_specific\,(d_g, i)$  // derive specific document model
8          update Index $i$ with $d_s$
9      end
10 end
```

Algorithm 3: Conceptual search pipeline at index-time.

```
1  $query\,(q)$
   Input: Query Document $q$, Maximum Result Size $k$, Indexes $I$, Ranker $O$
   Result: Relevant documents $R$
2  $R \leftarrow \emptyset$;
3  $f \leftarrow parse\,(q, P)$                // parse query format with Parser
4  $q_g \leftarrow map\_general\,(q, f)$     // map query to general document model
5  for $i \in I$ do
6      $q_s \leftarrow map\_specific\,(q_g, i)$      // derive specific document model
7      $r_t \leftarrow query\,(q_s, i)$          // get retrieval from specific index
8      $R \leftarrow R \cup r_t$                          // combine retrieval sets
9  end
10 $R \leftarrow sort(R, O)$          // assume ranking model & static features
11 $R \leftarrow \{R\,[0]\,, \ldots, R\,[k]\}$                  // prune ranked retrieval
12 return $R$
```

Algorithm 4: Conceptual search pipeline at query-time.

At query-time the pipeline is divided into two levels: retrieval and ranking. We do not impose which retrieval or ranking methods to use, but rather define necessary steps and components. For the retrieval-level, we discussed two retrieval methods, namely ranked Boolean retrieval and semantic search. As each retrieval set might have their own specific ordering, it becomes unclear how the combined result set should be sorted. Thus, we propose a ranking-level, that employs LTR to learn how to rank the retrieved documents automatically and in a way that minimizes an error function. In the context of ranking, we deploy similarity metrics that were introduced by the two aforementioned retrieval methods as features. Furthermore, we recommend the use of static features.

4 Experimental Evaluation

This chapter is about implementing the proposed search pipeline with the two retrieval methods: ranked Boolean and semantic search. Furthermore, we conduct experiments to evaluate their retrieval and ranking accuracy.
In Section 4.1 we present the datasets on which the experiments are conducted on. Next, in Section 4.2 we show our results for the respective approaches in the retrieval task. In Section 4.3 the ranking accuracy is evaluated. Finally, in Section 4.4 we discuss how various components in the pipeline were implemented.

4.1 Dataset

The conceptual pipeline is going to be evaluated by the following two datasets, which we are going to consider separately in the upcoming experiments.

4.1.1 TREC Washington Post Corpus

The evaluation mainly relies on the data that TREC provided for the News Track Background Linking Task from 2018-2020. This data is made up of two components: the TREC Washington Post Collection (WAPO) and corresponding test data, that is, human-rated relevance judgments for various test topics:

- News Track 2018: WAPO v2; test data: 50 topics, $8,508$ judgments in total
- News Track 2019: WAPO v2; test data: 57 topics, $15,655$ judgments in total
- News Track 2020: WAPO v3; test data: 49 topics, $17,764$ judgments in total

The human-rated relevance values map to the following judgments [52]:

- 0: The document provides little or no useful background information.
- 2: The document provides some useful background or contextual information that would help the user understand the broader story context of the target article.
- 4: The document provides significantly useful background ...
- 8: The document provides essential useful background ...
- 16: The document *must* appear in the sidebar otherwise critical context is missing.

The initial WAPO version (v1) contained duplicate entries with the same document identifier (*id* field in the JSON object.), which were removed in version 2. That version still contained a number of near-duplicate documents which have been removed in v3, with the help of a near-duplicate detection system. Additionally, v3 adds 154,418 new documents from 2018 and 2019, in comparison to v2. We choose to use WAPO v3 over previous versions, because it satisfies our assumption that there are close to no near-duplicate documents in the collection.

Originally, the WAPO v3 collection contains 671,947 news articles and blog posts from January 2012 through December 2019. We removed articles that are labeled in the *kicker* field as *Test*. These "test" articles only contain sample text, such as *"Lorem ipsum ..."*, and do not represent real articles. For this reason, they can be seen as noise. Furthermore, articles without a body are removed as well. In total we removed 4,179 articles.

The TREC news track in 2020 is based on the WAPO v3 collection, though previous tracks in 2018 and 2019 are based on version v2. We do not want to discard test data from previous tracks that contain articles only present in v2. At the same time, we want to use the superior version v3.

As a compromise, we added 2, 241 v2-articles that are referenced in previous test data and are not contained in v3 to the WAPO v3 collection. Finally, the cleaned and updated WAPO v3 collection contains **670,009** documents.

According to the TREC 2020 News Track Guidelines [52], articles from the dataset that are labeled in the *kicker* field as *Opinion*, *Letters to the Editor*, or *The Post's View*, are considered irrelevant by the human-raters. Initially, we made the mistake and removed these "irrelevant" articles as well, but later found out that they are referenced in the test data as negative examples. For this reason, it is not advisable to remove these articles and rather use the *kicker* field as a (negative) relevance signal for the ranking phase.

4.1.2 Netzpolitik.org

The motivation behind including another dataset for the evaluation is to have more data diversity. This comes with additional implementation efforts and require more hardware resources, mainly memory- and computational-costs. We choose the German news outlet *Netzpolitik.org* for the following reasons:

- as it is a German outlet, we can explore and evaluate our framework for the German language as well, instead of only the English language. This is going to reveal whether our concept is applicable for the German language, and which configurations would be necessary.
- all of its articles are online accessible, without charge or restrictions, e.g., pay-walls or limited usage.
- its simple and predictable URL routing patterns make crawling articles easier.
- most articles contain links to other internal articles that support the content. These internal references are considered as background links and are needed for the recall evaluation (see Section 4.2).
- most articles contain human-annotated keywords that sum up the content. These keywords can be used for retrieval processes.

For this outlet, we crawled all articles published from 2012-2020, which totals to **14,246** articles, ignoring articles without a title or body.

4.2 Retrieval Experiments

4.2.1 Recall Evaluation

With prioritizing recall, the ranking of the retrieved documents based on relevance is not important at this stage. In other words, we are not interested in how relevant (e.g., grades from 1-5) but whether the document is relevant or not (binary state), given a query. Thus binary judgment lists are needed for the evaluation of unranked retrieval sets.

For the WAPO corpus, we derive the binary judgment list from the combined scaled judgment lists from 2018-2020 (see Section 4.1.1), by simply defining a cut-off threshold for the grade, such that documents with relevance grades below the threshold are considered not relevant, whereas documents with grades above or equal to the threshold are considered relevant.

For the Netzpolitik.org corpus, there are no scaled judgment lists available. However, there are many other ways for generating a binary judgment list. Usually, clickstream logs of a live search engine are analyzed and used as a proxy for relevance. This requires a monitoring infrastructure and a user base, which is not feasible for now. We argue that manually placed, internal links by the author(s) refer to news articles, that are relevant to the given article, otherwise the author would not have included it. Thus, internal links are used as an heuristic for background links. Based on these links, we can automatically generate a binary judgment list for the Netzpolitik.org corpus, which is described in Algorithm 5.

*Note: Referenced articles that are not part of the current document collection of the given article are ignored, as $R \subseteq D$. Only **internal** links are considered.*

```
1  $get\_binary\_judgment\_list\,(D)$
   **Input:** Collection of Documents $D$
   **Result:** Binary Judgment List $L$
2  $L \leftarrow \emptyset$;
3  **for** $d \in D$ **do**
4      $K \leftarrow \emptyset$                    // dynamic list of relevant documents
5      $U \leftarrow get\_urls\,(d)$              // dynamic list of document links
6      **for** $u \in U$ **do**
7          $d_u \leftarrow get\_document\_by\_url\,(D, u)$;
8          **if** $d_u \in D$ **then**
9              $K \leftarrow K \cup \{d_u\}$;
10         **end**
11     **end**
12     **if** $|K| > 0$ **then**
13         update $L$ with $(d, K)$;
14     **end**
15 **end**
16 **return** $L$;
```

Algorithm 5: Function describing the generation of a binary judgment list that is based on manually placed, internal links in a document.

The TREC 2020 guideline actually advices against the use of links:

"Note that links already present in the Washington Post article collection are not training data for this task. In our conversations with the Post, their current practice is largely driven by the author of the article and does not follow any fixed guidelines or goal" [52].

This is understandable for the ranking task, due to the inaccuracy of the binary relevance judgments. However, for the retrieval task, this might be useful for evaluating the recall, especially, when human-annotated judgment lists are sparse and link-based binary judgment lists can be generated automatically.

The recall computation for different retrieval methods is described in Algorithm 6. In this context a retrieval method refers to a distinct indexer(-module)-index pair.

In the following three sections we are going to show our recall results for the ranked Boolean retrieval, semantic search and the combination of both retrieval methods on the WAPO- and Netzpolitik-corpus respectively. The retrieval recall is calculated based on Algorithm 6.

```
compute_average_recall ($L, C, M, I$)
Input: Binary Judgment List $L$, Document Cache $C$, Indexer Module $M$, Index $I$
Result: Average Recall $recall_{Average}$
$recall_{average} = 0$;
$Q_{ids} \leftarrow get\_query\_ids(L)$;
for $q_{id} \in Q_{ids}$ do
    $R_{relevant} \leftarrow get\_relevant\_documents(q_{id}, L)$;
    $q_{doc} \leftarrow get\_doc\_by\_id(q_{id}, C)$          // get general document model
    $q_I \leftarrow get\_query(q_{doc}, M)$          // get specific document model
    $R_{retrieval} \leftarrow get\_retrieval(q_I, I)$          // query specified index
    $recall_{retrieval} = 0$;
    for $r \in R_{retrieval}$ do
        if $r \in R_{relevant}$ then
            $recall_{retrieval} = recall_{retrieval} + \left(\frac{1}{|R_{relevant}|}\right)$;
        end
    end
    $recall_{average} = recall_{average} + \left(\frac{recall_{retrieval}}{|Q_{ids}|}\right)$;
end
return $recall_{average}$;
```

Algorithm 6: Function describing the recall computation, which is applicable for different retrieval methods.

4.2.2 Ranked Boolean Retrieval

Before we can run the retrieval experiments for the ranked Boolean retrieval, each document of both corpora needs to be represented as a set of keywords. We extract document keywords automatically based on the TF-IDF metric (see Section 2.1). The documents are then indexed in respective indexes, with regard to the corpus. Articles from the Netzpolitik-corpus contain keywords that are placed by the author and sum up the content - they allow us to compare human-annotated keywords vs. algorithmically extracted ones, with respect to the retrieval recall. In this context both types of keywords are used as a query to retrieve relevant documents from the respective indexes.

Model	$k = 100$	$k = 150$	$k = 200$	$k = 250$	$k = 300$
Human-annotated keywords	58.24%	62.49%	65.35%	67.51%	69.12%
Extracted keywords (TF-IDF)	63.67%	67.42%	70.21%	71.97%	73.51%
Combined (concatenated query)	67.90%	71.95%	74.52%	76.28%	77.66%
Combined (concatenated results)	65.99%	70.07%	72.91%	74.89%	76.34%

Table 4.1: Retrieval recall of the ranked Boolean model for the Netzpolitik.org corpus.

Table 4.1 shows the recall results for the Netzpolitik-corpus for different maximum retrieval sizes k. We combine both keyword types for retrieval in two ways:

- The union of both keyword sets form the combined query (concatenated query) that is used to retrieve relevant documents.
- The union of the respective retrieval sets for each keyword type. The maximum retrieval size for each keyword type is $\frac{k}{2}$, such that the combined retrieval size does not exceed k.

The combined retrieval, that is, the concatenated query of both keyword types, perform best among the others with a recall value ranging from 67.90% to 76.66%, depending on k. It is interesting to see that algorithmically extracted keywords outperform human-annotated ones by at least 4.39% and up to 5.43%.

This makes sense, considering that each document is indexed by its algorithmically extracted keywords (based on TF-IDF). One could argue that human-annotated keywords would perform better if each document was indexed by its human-annotated keywords instead. It cannot be guaranteed that human-annotated keywords are present in a given document, which is why it is preferable to not rely on this data. However, if present, human-annotated keywords showed to improve recall by approximately 4%, comparing the combined (concatenated query) with the extracted keywords (TF-IDF) retrieval recall.
Table 4.2 shows the recall values for the WAPO-corpus that range from 57.17% to 74.49%, which is rather similar in comparison to the equivalent query type for the Netzpolitik-corpus, which ranges from 63.67% to 73.51%. It should be highlighted that there is a similarity in recall, even though the binary judgment lists for both corpora are generated differently (see Section 4.2.1). This indicates that internal links can be used as a heuristic for background links in the context of retrieval recall.

Model	$k = 100$	$k = 150$	$k = 200$	$k = 250$	$k = 300$
Extracted keywords (TF-IDF)	57.17%	63.72%	68.24%	71.62%	74.49%

Table 4.2: Retrieval recall of the ranked Boolean model for the WAPO corpus.

4.2.3 Semantic Search

Our proposed semantic search approach is based on sentence embeddings (see Section 3.4.2). We choose characteristic text components in sentence length that approximately represent the underlying document, on which sentence embeddings can be calculated:

- title
- first paragraph
- section titles
- human-annotated keywords

- algorithmically extracted keywords

The text components: title, first paragraph and section titles are embed as is, without further preprocessing. Text components based on keywords are simply concatenated to a string, delimited by whitespaces.

We differentiate algorithmically extracted keywords in:

- TF-IDF normalized
- TF-IDF denormalized
- TF-IDF denormalized, order preserved

For the efficient extraction of TF-IDF based keywords we rely on an inverted index that requires its index- and query-terms to be normalized (see Section 2.1). In our implementation the normalization process is based on stemming instead of lemmatization, thus stemmed terms may not correspond to a "real" word that could be found in a dictionary. This is necessary in the context of searching an inverted index, as normalized, stemmed words relax strict matching rules, such that more results can be found. However, stemmed words that do not correspond to real words anymore might lose their semantics or meaning. Furthermore, even if words are lemmatized, inflections might carry meaning as well, which would be lost.
We anticipated this problem and propose a method to recreate normalized words, which we refer to as **denormalization**. A normalized keyword and the document text, from which it was extracted, are used as the input for our denormalization method, which is described in Algorithm 7. The output, that is, the denormalized word, should roughly correspond to the originally extracted keyword (before normalization), assuming the normalized keyword was extracted from the provided document text. With this assumption and further assuming that normalized words are stemmed not lemmatized, it is guaranteed to find a corresponding denormalized word.

```
1  $denormalize\,(t, kw)$
   Input: Document Text $t$, Keyword $kw$
   Result: Denormalized Keyword $kw_{denorm}$
2  $kw_{temp} \leftarrow kw$;
   // match the first word in the text containing the substring
3  $kw_{denorm} \leftarrow match\_substring\,(t, kw_{temp})$;
4  while $kw_{denorm} = \emptyset$ do
5      $kw_{temp} \leftarrow drop\_last\_character\,(k_{temp})$;
6      $kw_{denorm} \leftarrow match\_substring\,(t, kw_{temp})$;
7      if $length\,(kw_{temp}) <= 1$ and $kw_{denorm} = \emptyset$ then
8          return $\emptyset$;
9      end
10 end
11 return $kw_{denorm}$;
```

Algorithm 7: Function describing the denormalization of extracted TF-IDF keywords, given the text that they were extracted from.

Grammatical order is crucial for semantics, thus it make sense to preserve the order of keywords that appear in document. For the implementation we store the position in the underlying document text for each index-term in the aforementioned inverted index. This enables us to preserve the grammatical order of extracted keywords, which is considered when concatenating the keywords to a string.

Before index- and query-time, there is a need for extracting aforementioned text components. This requires a specific Parser-module (see Figure 3.2) for each document collection, as their original document structure varies. The Parser-component converts documents from each collection to the general document model (see Section 3.2) and passes them to the Indexer-component as the input. Then, text components are extracted and embed by an Indexer-module. The resulting embeddings are subsequently used for indexing and querying a vector index that is based on HNSW graphs (see Section 2.2.7) in the Indexes-component. For both data collections we choose n prospect text components and evaluate every n^2 combination in a query-index matrix, in which rows represent the text components at query-time and columns represent them at index-time.

	title	title & first paragr.	title & section titles	keyw. annot.	keyw. tf-idf	keyw. tf-idf, denorm.	keyw. tf-idf, denorm. ord.
title	23.74%	20.73%	21.88%	14.27%	1.86%	17.81%	18.08%
title & first paragr.	19.27%	34.63%	14.68%	10.48%	3.91%	16.95%	18.08%
title & section titles	21.37%	20.22%	22.33%	13.73%	2.54%	18.50%	19.08%
keyw. annot.	12.14%	9.33%	12.22%	36.24%	7.32%	18.89%	19.08%
keyw. tf-idf	1.26%	1.37%	1.70%	2.78%	20.61%	3.38%	3.67%
keyw. tf-idf, denorm	12.14%	8.84%	12.48%	14.23%	5.71%	30.06%	29.59%
keyw. tf-idf, denorm. ord.	13.18%	10.86%	13.40%	14.82%	5.64%	30.42%	30.60%

Table 4.3: Retrieval recall of the semantic search approach with a maximum retrieval size $k = 100$ for the Netzpolitik.org corpus.

	title	title & first paragr.	title & section titles	keyw. tf-idf	keyw. tf-idf, denorm.	keyw. tf-idf, denorm. ord.
title	13.08%	13.64%	13.24%	10.82%	13.09%	15.79%
title & first paragr.	11.13%	15.14%	11.00%	11.99%	14.88%	17.50%
title & section titles	13.17%	13.73%	13.29%	10.44%	12.93%	15.80%
keyw. tf-idf	7.46%	8.43%	7.17%	20.79%	18.26%	19.26%
keyw. tf-idf, denorm	10.34%	12.15%	10.62%	19.34%	24.06%	26.28%
keyw. tf-idf, denorm. ord.	11.96%	14.63%	11.66%	20.52%	26.24%	28.58%

Table 4.4: Retrieval recall of the semantic search approach with a maximum retrieval size $k = 100$ for the WAPO corpus.

Table 4.3 shows the semantic search recall values for the Netzpolitik corpus with a maximum retrieval count $k = 100$. The recall values range from 1.26% to 36.24%, showing a wide spread for different text component pairings. The highest recall is achieved by human-annotated keywords at query- and index-time with 36.24%. Algorithmically extracted keywords perform worse with up to 30.60% recall. This indicates that human-annotated keywords carry more meaning than algorithmically extracted ones, which are solely based on occurrence statistics, and are just a heuristic for semantics. The second highest recall is achieved by the combined text components *title & first paragraph* with 34.63%. This could be due to the fact that the first paragraph typically summarizes the main story of the news article and thus represents a document's semantics well. At the third place are the denormalized and order-preserved keywords at index- and query-time, achieving 30.60% recall and performing best among the algorithmically extracted keywords. This validates the assumptions we made earlier about denormalizing keywords and keeping their grammatical order to improve semantics and thus recall.

Table 4.4 shows the semantic search recall results for the WAPO corpus with a maximum retrieval count $k = 100$. It can be immediately seen that keywords at index- and query-time perform best, ranging from 18.26% and up to 28.58%. Again, from the algorithmically extracted keywords, the denormalized ones, with the grammatical order preserved, perform best, confirming the importance of denormalization and grammatical order for semantics.

It can be observed that the recall results for both news outlets can differ drastically, e.g., the combined text components *title & first paragraph* at query- and index-time perform more than twice as well (34.63%) for the Netzpolitik corpus in comparison to the WAPO corpus (15.14%). This shows how domain-specific the search problem is and that there is no single "silver bullet" solution. Each domain might has its own unique definition of relevance and thus needs to be evaluated separately [22, p.4-6].

4.2.4 Combined Retrieval

In Section 3.1 we propose a IR framework in which it is possible to deploy multiple retrieval methods in parallel and combine their results. This comes with the rationale that in order to maximize recall, it is beneficial to combine various retrieval methods, instead of relying on a single one.
The highest recall results for semantic search (see Section 3.4.2) are substantially lower, almost half the value, compared to the recall of the ranked Boolean retrieval (see Section 4.2.2). However, even though the recall for semantic search is comparably low, in combination with the ranked Boolean model, it could introduce new articles that the baseline retrieval did not find, which would drastically improve overall recall.
The results in the previous sections show which configurations for both retrieval methods, that is, ranked Boolean and semantic search, achieve high recall for the respective corpus. By combining well-performing configurations for both retrieval methods we can evaluate to what extent semantic search can improve the baseline retrieval method.

Retrieval Metric	$k = 200$	$k = 250$	$k = 300$	$k = 350$	$k = 400$
Average recall	65.69%	67.92%	69.56%	71.09%	72.28%
Average recall improvement with Semantic Search	2.04%	2.09%	2.17%	2.18%	2.09%
Average number of newly introduced documents with Semantic Search	0.06	0.02	0.06	0.07	0.07

Table 4.5: Retrieval recall of the combined ranked Boolean model and semantic search approach for the Netzpolitik.org corpus.

Table 4.5 shows the combined retrieval for the Netzpolitik corpus, which is a combination of:

- Ranked Boolean Retrieval based on extracted TF-IDF keywords.
- Semantic Search based on extracted TF-IDF keywords at index- and query-time that are denormalized and ordered according to their grammatical order.

It should be highlighted that we did not choose the best performing configurations for the combined retrieval, but good performing ones, which also yield high recall for the WAPO corpus. This way, we can compare different datasets on the same configuration. It is apparent that there is only a small contribution to the recall performance with semantic search of about 2% for the observed maximum retrieval counts k.

Retrieval Metric	$k = 200$	$k = 250$	$k = 300$	$k = 350$	$k = 400$
Average recall	61.69%	65.44%	68.38%	70.39%	72.59%
Average recall improvement with Semantic Search	4.52%	4.54%	4.66%	4.22%	4.34%
Average number of newly introduced documents with Semantic Search	2.34	2.44	2.44	2.28	2.23

Table 4.6: Retrieval recall of the combined ranked Boolean model and semantic search approach for the WAPO corpus.

Table 4.6 depicts the combined retrieval recall for the WAPO corpus, which has the same combination of retrieval methods like the Netzpolitik corpus in Table 4.5.

It shows a small increase of about 4% in recall for various retrieval sizes k, as the average number of newly introduced documents with semantic search is approximately 2. This is a significant improvement over the values exhibited in Table 4.5.

Nevertheless, Table 4.5 and Table 4.6 show that semantic search is not beneficial for the retrieval task. For the same maximum retrieval size k, a higher recall could be achieved using the baseline retrieval method exclusively, that is, ranked Boolean retrieval.

4.3 Ranking Experiments

In the previous section the results show that semantic search is not effective for the retrieval task, not even as a complementary approach, as the baseline retrieval method is superior. In this section we are going to evaluate the ranking accuracy of both retrieval methods, which is going to reveal whether semantic search is effective for the ranking task or not.

For the ranking evaluation we are only going to consider the WAPO corpus, as it comes with human-rated and scaled judgment lists that are required for evaluation. We do not have human-rated judgment lists for the Netzpolitik corpus. Furthermore, we are going to ensure for each experiment that the query article d_i is not contained in the result set R, as $d_i \notin R$ has to hold (see Section 3.1). Depending on the context, the result set could be the set of retrieved documents or the set of relevance judgments, with respect to a query document.

4.3.1 Assuming optimal Retrieval

For this experiment we assume a "perfect" retrieval, that is, optimal recall and optimal precision, with respect to the judgment list. This way, we can better differentiate the ranking accuracy from the retrieval performance of the underlying IR approach. An optimal recall can be achieved by working on the judgment lists directly. Furthermore, to guarantee optimal precision, we filter out irrelevant judgments that have a relevance score of 0.

Subsequently, without the need for a retrieval step, a feature of interest is extracted for each judgment in the corresponding judgment list, given a query. There is no need for a supervised ranking model, as we consider one feature at a time and can simply rank the judgments by the considered feature. For the BM25 score and cosine similarity the ordering should be descending: the higher each metric, the higher the corresponding lexical or semantic similarity.

Model	Judgment List 2018 with $k = 170 \approx \frac{8{,}508\ judgments}{50\ topics}$		Judgment List 2019 with $k = 275 \approx \frac{15{,}655\ judgments}{57\ topics}$		Judgment List 2020 with $k = 362 \approx \frac{17{,}764\ judgments}{49\ topics}$	
	nDCG@5	nDCG@10	nDCG@5	nDCG@10	nDCG@5	nDCG@10
Baseline (BM25 score)	0.6408	0.7074	0.6828	0.7173	0.7268	0.7594
Semantic Search (Cos. Sim.)	0.6322	0.6968	0.6167	0.6701	0.6615	0.6974

Table 4.7: Ranking accuracies of individual IR methods for the WAPO Corpus, assuming optimal recall and precision.

Table 4.7 shows the ranking results for two features introduced by the ranked Boolean model and semantic search, that is, BM25 score and cosine similarity, for respective judgment lists, assuming optimal retrieval. The ranking accuracy of the baseline approach with its BM25 feature outperforms the semantic search model slightly to moderately for both *nDCG*-metrics.
The experiment shows that on its own, the cosine similarity feature introduced by semantic search is an inferior relevance signal in comparison to the BM25 score feature introduced by the ranked Boolean model.

4.3.2 Actual Retrieval

In the following experiments we are going to test the actual retrieval sets of various models on the WAPO judgment list 2020.

Model	$k = 100$		$k = 150$		$k = 200$	
	nDCG @5	nDCG @10	nDCG @5	nDCG @10	nDCG @5	nDCG @10
Baseline (BM25 score)	0.5205	0.5198	0.5205	0.5198	0.5205	0.5198
Semantic Search (Cos. Sim.)	0.3255	0.2922	0.3255	0.2922	0.3255	0.2927

Table 4.8: Actual ranking accuracy of individual IR methods for different maximum retrieval sizes k, tested on the WAPO 2020 judgment list.

Table 4.8 shows the actual ranking accuracy for both IR methods. Again, the baseline model is superior to the semantic search method. There is a significant difference in ranking accuracy for the optimal vs. actual retrieval, especially for the semantic search method that can achieve more than double the nDCG score under optimal retrieval conditions. This indicates that the retrieval step plays an important role for the ranking accuracy. The spread in ranking accuracy is smaller for the baseline approach, suggesting a superior retrieval to semantic search, that is, better recall and/or precision.

To this end we considered aforementioned retrieval methods in isolation. In the following we are going to combine both approaches and additionally introduce two static features:

- The retrieved document's length, that is, the character count.
- A binary value that represents whether the query document was published after the retrieved document or not, which is indicated by a corresponding 1 or 0. This time feature could be a strong relevance signal, as background links usually point to the past [53].

To handle more than one feature, we deploy the supervised ranking model LambdaMART (see Section 2.2.6) for the ranking task, which is trained on the 2018 and 2019 judgment lists and tested on the 2020 judgment list.

Model	$k = 100$		$k = 150$		$k = 200$	
	nDCG @5	nDCG @10	nDCG @5	nDCG @10	nDCG @5	nDCG @10
Baseline & Semantic Search	0.3324	0.3702	0.3212	0.3590	0.3118	0.3519
Baseline & Doc. length	0.2539	0.3008	0.2190	0.2778	0.2035	0.2593
Baseline & Time	0.3270	0.3370	0.3036	0.3174	0.2915	0.3083
All features combined	0.3314	0.3579	0.2951	0.3350	0.2822	0.3287

Table 4.9: Ranking accuracy of various feature combinations, tested on the WAPO 2020 judgment list.

Even though we combine both retrieval approaches, that is, union their retrieval sets, and additionally introduce static features, Table 4.9 shows that the ranking accuracies barely increase or even decrease in comparison to the individual retrieval methods, without a supervised model. There are several plausible explanations for this:

- The supervised ranking model is not well-suited for a handful of features.
- There is not enough training data for the supervised model.
- The hyperparameters for the supervised model are not appropriate.
- All considered features, except the BM25 score, are actually detrimental for ranking.

In the end, the baseline retrieval method with its BM25 score achieves the highest nDCG scores and thus the best ranking accuracy in the context of the background linking task.

4.4 Implementation

In this section we elaborate our implementation choices for the proposed search pipeline.

All conducted experiments can be reproduced, as we published the source code of the implementation on Github:

https://github.com/DucAnhPhi/NewsSearchEngine

This project is implemented in Python (3.6.9) and is tested on Ubuntu 18.04.2 LTS.

4.4.1 Web Scraping Netzpolitik.org

"[...] web scraping is the practice of gathering data through any means other than a program interacting with an API (or, obviously, through a human using a web browser). This is most commonly accomplished by writing an automated program that queries a web server, requests data (usually in the form of HTML and other files that compose web pages), and then parses that data to extract needed information" [54].

For generating the Netzpolitik.org corpus we implemented a python script that scrapes news articles in the form of HTML files from 2012 to 2020. Furthermore, we implemented unit tests for each of the 9 years, to make sure that the data is parsed and extracted correctly, as HTML structures are usually subject to change in the real world. These unit tests can be seen as data quality assurance and are crucial for the search pipeline that goes after the garbage-in-garbage-out principle. Even though this project is implemented in Python (3.6.9) the large size of HTML test files for the corresponding unit tests make up about 80% of the source code.

4.4.2 Embedding Models for Semantic Search

The semantic search retrieval is based on sentence embeddings that require trained embedding models (see Section 2.1.7). In this work we resort to pre-trained and open-sourced embedding models that are presented in the following.

For the Netzpolitik.org corpus, which is a German news outlet, we deployed the German BERT model **bert-based-german-cased** provided by:
https://deepset.ai/german-bert.
The German BERT model outperforms multilingual models that are open-sourced by Google. It is trained on 6 GB of German Wikipedia text, 2.4 GB of the German legal text, and 3.6 GB of German news articles.

For the WAPO corpus, which is an American news outlet based on the English language, we deploy the BERT model **stsb-disilbert-base** provided by:
https://www.sbert.net/docs/pretrained_models.html.
The pre-trained model is optimized for evaluating similarity between embeddings, known as *Semantic Textual Similarity*(STS). It is trained on the following datasets:

- **Stanford Natural Language Inference Corpus** (SNLI): comprises of 570k human-written English sentence pairs that are manually labeled with *entailment*, *contradiction*, and *neutral*, supporting the task of recognizing textual entailment.
- **Multi-Genre Natural Language Inference Corpus** (MultiNLI): comprises of 433k annotated sentence pairs that are similar to the SNLI corpus. MultiNLI differs from SNLI in that it covers a range of genres of spoken and written text.

Furthermore, the model is fine-tuned on the STS benchmark train set, which comprises of a selection of English datasets that are used for evaluating STS tasks.

4.4.3 Vector Storage for Semantic Search

We deploy HNSW graphs for the index structure of the semantic search retrieval (see Section 3.4.2).

The underlying paper by [31] comes with a corresponding header-only *C++* implementation with Python bindings, which we used in this work. The paper's code is available at:

https://github.com/nmslib/hnswlib.

4.4.4 Document Cache, Inverted Index and NLP

Elasticsearch is a distributed, free and open search and analytics engine for all types of data [...]. [25]

Furthermore, Elasticsearch provides an efficient and feature-rich API for indexing and querying documents.

In this work we deploy **Elasticsearch** extensively for various tasks:

- The Document Cache and the inverted index for the corresponding corpora are implemented as a single Elasticsearch index, which is essentially an inverted index that comes with a lot of features.
- We extensively make use of its *Term vectors API* for effectively retrieving top-k TF-IDF keywords for a given document. More specifically, we retrieve up to three keywords from a document's title and up to 25 keywords for a document's content.
- For the indexing of documents, Elasticsearch comes with pre-defined *language analyzers* that are *"aimed at analyzing specific language text"* [55], that is, automatically removing stopwords, tokenizing and normalizing the remaining terms that subsequently make up the index-terms for the inverted index. We mostly used the pre-defined analyzers, that is, the *english* and *german* analyzers for the corresponding corpus. The only modification we made, is adding more elaborate stopword lists for each of both mentioned analyzers, to filter out more German and English stopwords respectively.

Our Elasticsearch configurations can be found in the aforementioned Github repository.

4.4.5 LambdaMART

For the LambdaMART implementation we resort to the **LightGBM** Python package, which is a "gradient boosting framework that uses tree based learning algorithms", and is available on Github:

https://github.com/microsoft/LightGBM

More specifically, we make use of LightGBM's LGBMRanker class, with its default hyperparameter settings, for our supervised ranking model. More detailed information can be found at:

https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRanker.html

4.4.6 Ranking Evaluation

For computing *nDCG*-metrics for the ranking evaluation we make use of the evaluation tool **trec_eval**, which is provided by TREC and is also available on Github:

https://github.com/usnistgov/trec_eval

5 Conclusion and Future Work

In the following we are going to summarize our work. Finally, the results are discussed with regard to future work.

5.1 Summary

We reformulated the background linking task as a search problem that is divided into two subproblems: retrieval and ranking. Based on these two processes we modeled a conceptual search pipeline that can be seen as an IR framework, as it gives a guideline of which components are needed for search, without requiring specific IR methods. This modularity allows for tailoring the search pipeline according to domain-specific needs and is not limited to the background linking task. Furthermore, this framework supports the deployment of various IR methods in parallel and combines their retrieval results. Although the recall can only increase with the union of corresponding retrieval sets, the precision and subsequently the ranking accuracy could decrease. It still remains an open question how to balance recall and precision to maximize ranking accuracy.

Following the proposed framework, we discussed and deployed two drastically different IR methods, that is, ranked Boolean and semantic search. We then compared and evaluated their retrieval recall and ranking accuracy on two document collections:

- English-speaking TREC Washington Post Corpus provided by TREC for its News Track Background Linking Task from 2018-2020 (see Section 4.1.1).
- German Netzpolitik.org articles that were published from 2012-2020 (see Section 4.1.2).

With this corpus we could show that manually placed internal links in news articles can be used as an heuristic for background links, which is useful for evaluating retrieval recall (see Section 4.2.2).

Experiments were conducted to find the best configurations for the semantic search approach, that is, suitable text for corresponding sentence embeddings were identified at index- and query-time to maximize recall (see Section 4.2.3).
Even with optimized sentence embedding for semantic search, the experiments revealed that the approach is not suitable for the retrieval as well as ranking task:

- For the retrieval task, higher recall values can be achieved for the same retrieval size k by solely deploying the ranked Boolean model.
- Similarly, for the ranking task, higher ranking accuracy can be achieved using just the baseline method, namely the ranked Boolean model.

For the ranking task we additionally introduced two static features, the document length and a binary value, representing whether the query documents was published after the retrieved document. Experimental results show that these static features were insignificant for the ranking accuracy or even detrimental.

Even though we showed that the deployed semantic search approach is not effective, we want to highlight that our proposed framework, the conceptual search pipeline, is still a promising model for solving the background linking task, with its versatility, as IR methods can be used in parallel or replaced, and practical considerations for real world applications. We argue that it is beneficial to deploy various IR models in line with our framework, instead of relying on a single IR model: With the increasing diversity in the retrieval results our multi-model system could find background links that a single-model system might miss. Furthermore, multiple IR methods introduce several features that could be beneficial for the ranking task.

5.2 Discussion and Future Work

In this work, we considered IR methods that try to find the most similar documents to a given query document, in terms of lexical overlap or semantics, under the assumption that most duplicate or near-duplicate documents are removed from the considered document collection (see Section 3.1). This way, similarity can be used as a heuristic for background links. However, this assumption is not always feasible in practice and comes with caveats.
Instead of finding the most similar document to a given query, it makes sense to find documents that share "some" common ground with the query, and add new or more detailed information about the news story that provides important context or background information to the reader. It is difficult to define the loose notions of "some" and "new or more detailed information", thus it is not clear at this point how to model documents accordingly. However, the use of graph-based approaches that rely on named-entities and knowledge graphs are a promising direction for future work.

To this end, the optimal recall to precision ratio for maximizing ranking accuracy remains an open question. The optimal ratio can differ for different domains and IR methods, though it would be interesting to investigate general recall/precision boundaries that are applicable across different IR methods and domains.

For the ranking task experimental results show that the combination of multiple features yield worse results than using just the baseline method. Further work needs to investigate whether the considered features are actually detrimental for ranking or perform worse due to other factors such as a lack of hyperparameter tuning, a lack of training data for the supervised model, or choosing an inappropriate ranking model. Furthermore, special attention should be drawn to finding effective static features, which require less computation.

Finally, it should be highlighted that our baseline approach achieves an nDCG@5 score of **0.5205** on the TREC 2020 test data. In comparison, the best performing model of the TREC News Track 2020 (that was published at the time of this writing [56]) is the approach by Khloponin et al. [41] that achieves an nDCG@5 score of **0.5924**. Similar to our baseline method, it is based on the ranked Boolean model that relies on the Okapi BM25 measure. This indicates that there is still room for improvement for the research community, as a ranked Boolean model is not providing the ideal ranking for background links (even assuming perfect recall), and misses some relevant documents in the retrieval step entirely.

Acronyms

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
DCG	Discounted Cumulative Gain
FFN	Fully Connected Feed-Forward Network
HNSW	Hierarchical Navigable Small World
HTML	Hypertext Markup Language
ID	Identifier
IR	Information Retrieval
IT	Information Technology
JSON	JavaScript Object Notation
K-ANNS	K-Approximate Nearest Neighbor Search
K-NNS	K-Nearest Neighbor Search
LSI	Latent Semantic Indexing
LTR	Learning To Rank
MART	Multiple Additive Regression Trees

MLM	Masked Language Modeling
MSE	Mean Squared Error
MultiNLI	Multi-Genre Natural Language Inference
NDCG	Normalized Discounted Cumulative Gain
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NSP	Next Sentence Prediction
RAKE	Rapid Automatic Keyword Extraction
S-BERT	Sentence-BERT
SNLI	Stanford Natural Language Inference
STS	Semantic Textual Similarity
SVD	Singular Value Decomposition
TREC	Text Retrieval Conference
UML	Unified Modeling Language
WAPO	Washington Post
XML	Extensible Markup Language

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] News in the u.s. - statistics & facts. `https://www.statista.com/topics/1640/news/`, 2020. Accessed: 2021-04-25.

[2] I. Soboroff, S. Huang, and D. Harman. Trec 2018 news track overview. In *TREC*, 2018.

[3] Marwa Essam and Tamer Elsayed. Why is that a background article: A qualitative analysis of relevance for news background linking. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, pages 2009–2012, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[5] Dipanjan Sarkar. *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from Your Data.* Springer, 2016.

[6] Micheal W. Berry and Jacob Kogan. *Text Mining: Applications and Theory.* John Wiley & Sons, Ltd, 2010.

[7] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 216–223, USA, 2003. Association for Computational Linguistics.

[8] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text Mining: Applications and Theory*, pages 1 – 20, 03 2010.

[9] What is the softmax function? `https://deepai.org/machine-learning-glossary-and-terms/softmax-layer`, 2020. Accessed: 2021-04-28.

[10] Loss functions. `https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html`, 2020. Accessed: 2021-04-28.

[11] The illustrated transformer. `https://jalammar.github.io/illustrated-transformer/`, 2018. Accessed: 2021-04-30.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762, June 2017.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv e-prints*, page arXiv:1607.06450, July 2016.

[14] The illustrated bert, elmo, and co. (how nlp cracked transfer learning). `http://jalammar.github.io/illustrated-bert/`, 2018. Accessed: 2021-04-29.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, page arXiv:1810.04805, October 2018.

[16] The bidirectional language model. `https://medium.com/@plusepsilon/the-bidirectional-language-model-1f3961d1fb27`, 2018. Accessed: 2021-04-29.

[17] How the embedding layers in bert were implemented. `https://medium.com/@_init_/why-bert-has-3-embedding-layers-and-their-implementation-details-9c261108e28a`, 2019. Accessed: 2021-04-29.

[18] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes,

and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv e-prints*, page arXiv:1609.08144, September 2016.

[19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019.

[20] Davide Chicco. *Siamese Neural Networks: An Overview*, pages 73–94. Springer US, New York, NY, 2021.

[21] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. *arXiv e-prints*, page arXiv:1412.6622, December 2014.

[22] Doug Turnbull and John Berryman. *Relevant Search: With applications for Solr and Elasticsearch.* Manning Publications Co., 2016.

[23] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 426–434, New York, NY, USA, 2003. Association for Computing Machinery.

[24] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definite Guide.* O'Reilly Media, Inc., 2015.

[25] What is elasticsearch? `https://www.elastic.co/what-is/elasticsearch`, 2020. Accessed: 2021-03-17.

[26] Elasticsearch learning to rank. `https://elasticsearch-learning-to-rank.readthedocs.io/en/latest/core-concepts.html`, 2017. Accessed: 2021-03-19.

[27] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 01 2009.

[28] Practical bm25 - part2: The bm25 algorithm and its variables. `https://www.elastic.co/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables`, 2018. Accessed: 2021-04-27.

[29] Christopher Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11, 01 2010.

[30] Lambdamart overview. `https://wellecks.wordpress.com/tag/lambdamart/`, 2015. Accessed: 2021-05-02.

[31] Yury A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016.

[32] Economic impact assessment of nist's text retrival conference (trec) program. `https://trec.nist.gov/pubs/2010.economic.impact.pdf`, 2010. Accessed: 2021-04-26.

[33] Trec homepage. `https://trec.nist.gov/`, 2021. Accessed: 2021-03-13.

[34] Trec news track. `http://trec-news.org/`, 2021. Accessed: 2021-03-08.

[35] Shahrzad Naseri, John Foley, and James Allan. Umass at trec 2018: Car, common core and news tracks. In *TREC*, 2018.

[36] Marwa Essam and Tamer Elsayed. bigir at trec 2019: Graph-based analysis for news background linking. In *TREC*, 2019.

[37] Pepijn Boers, Chris Kamphuis, and Arjen de Vries. Radboud university at trec 2020. In *TREC*, 2020.

[38] Anand Deshmukh and Sethi Udhav. IR-BERT: Leveraging BERT for Semantic Search in Background Linking for News Articles. *arXiv e-prints*, page arXiv:2007.12603, July 2020.

[39] Agra Bimantara, Michelle Blau, Kevin Engelhardt, Johannes Gerwert, Tobias Gottschalk, Philipp Lukosz, Shenna Piri, Nima Saken Shaft, and Klaus Berberich. htw saar @ trec 2018 news track. In *TREC*, 2018.

[40] Kuang Lu and Hui Fang. Leveraging entities in background document retrieval for news articles. In *TREC*, 2019.

[41] Pavel Khloponin and Leila Kosseim. The clac system at the trec 2020 news track. In *TREC*, 2020.

[42] Divide-and-conquer paradigm. `https://silo.tips/download/divide-and-conquer-paradigm`, 2016. Accessed: 2021-03-08.

[43] Divide-and-conquer algorithms. `https://en.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms`, 2021. Accessed: 2021-03-08.

[44] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 105–114, New York, NY, USA, 2011. Association for Computing Machinery.

[45] Andreas Spitz and Michael Gertz. Terms over load: Leveraging named entities for cross-document extraction and summarization of events. SIGIR '16, pages 503–512, New York, NY, USA, 2016. Association for Computing Machinery.

[46] S. Nunes. State of the art in web information retrieval. Technical Report. FEUP, 2006.

[47] Yogesh Gupta, Ashish Saini, and A.K. Saxena. A review on important aspects of information retrieval. *International Journal of Computer and Information Engineering*, 7(12):1638–1646, 2013.

[48] Semantic search. `https://towardsdatascience.com/semantic-search-73fa1177548f`, 2019. Accessed: 2021-03-26.

[49] Kuang Lu and Hui Fang. Paragraph as lead - finding background documents for news articles. In *TREC*, 2018.

[50] Stanford lecture slides: Introduction to information retrieval. `https://web.stanford.edu/class/cs276/handouts/lecture14-learning-ranking.pdf`, 2019. Accessed: 2021-03-19.

[51] Hang LI. A short introduction to learning to rank. *IEICE Transactions on Information and Systems*, E94.D(10):1854–1862, 2011.

[52] Trec2020 guidelines. `http://trec-news.org/guidelines-2020.pdf`, 2020. Accessed: 2021-03-17.

[53] John Foley. Trec News Background-Linking 2018: Filter By Time! . `https://jjfoley.me/2019/07/24/trec-news-bm25.html`, 2019.

[54] Ryan Mitchell. *Web Scraping with Python*. O'Reilly Media,Inc.,1005 Gravenstein Highway North,Sebastopol,CA95472, 2018.

[55] Language analyzers. `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-lang-analyzer.html`, 2020. Accessed: 2021-04-21.

[56] Trec 2020 news track background linking task results. `https://trec.nist.gov/pubs/trec29/appendices/newsBackground.html`, 2020. Accessed: 2021-04-23.