ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

School of Information Sciences and Technology
Department of Informatics
Athens, Greece

Bachelor Thesis
in
Computer Science

QUERY EXPANSION TECHNIQUES AND DOCUMENT RE-RANKING FOR BIOMEDICAL INFORMATION RETRIEVAL

Vasilis Ioannidis

Supervisors: Dr. Antonia Kyriakopoulou
Department of Informatics
Athens University of Economics and Business

Dr. Katia Papakonstantinopoulou
Department of Informatics
Athens University of Economics and Business

February, 2024

Query Expansion Techniques and Document Re-Ranking for Biomedical Information Retrieval

by Vasilis Ioannidis

February 2024

Supervisors: Dr. Antonia Kyriakopoulou, Dr. Katia Papakonstantinopoulou

Athens University of Economics and Business

School of Information Sciences and Technology

Department of Informatics

Athens, Greece

Abstract

In this thesis, we investigate methods for improving information retrieval systems. In particular, we explore a plethora of query expansion techniques using vectors, statistics and Transformers. We expand the user's query with synonyms words and retrieve the top N documents. The presented models are tested on data from the TREC-COVID challenge which contains a collection of biomedical literature articles. Searching for relevant documents in such dataset is often a time-consuming process which leads to making a lot of searches before finding the right combination of words. For that issue we have tried to create models that will solve the above problem saving time from users-scientists. The methods that were used contribute to significant improvements in information retrieval accuracy compared to the classic BM25 method.

Περίληψη

Στην πτυχιακή αυτή , εξετάζουμε μεθόδους για τη βελτίωση των συστημάτων ανάκτησης πληροφοριών. Συγκεκριμένα, εξετάζουμε μια πληθώρα τεχνικών επέκτασης ερωτήσεων χρησιμοποιώντας διανύσματα, στατιστικά και Transformers. Επεκτείνουμε την ερώτηση του χρήστη με συνώνυμες λέξεις και ανακτούμε τα κορυφαία Ν έγγραφα. Τα παρουσιαζόμενα μοντέλα δοκιμάζονται σε δεδομένα από τον διαγωνισμό TREC-COVID που περιλαμβάνει μια συλλογή από άρθρα βιοϊατρικής βιβλιογραφίας. Η αναζήτηση για σχετικά έγγραφα σε τέτοια σύνολα δεδομένων είναι συχνά μια χρονοβόρα διαδικασία που οδηγεί στο να γίνονται πολλές αναζητήσεις πριν βρεθεί η σωστή συνδυασμός λέξεων. Για αυτό το πρόβλημα προσπαθήσαμε να δημιουργήσουμε μοντέλα που θα επιλύσουν το παραπάνω πρόβλημα, εξοικονομώντας χρόνο από τους χρήστες-επιστήμονες. Οι μέθοδοι που χρησιμοποιήθηκαν συνεισφέρουν σημαντικές βελτιώσεις στην ακρίβεια ανάκτησης πληροφοριών σε σύγκριση με την κλασική μέθοδο BM25.

Table of Contents

Acknowledgements

I would like to express my gratitude to Dr. Antonia Kyriakopoulou for her invaluable guidance and support throughout the duration of my thesis. Her expertise and feedback have been instrumental in shaping the direction of my research and its development and success.

Also I would like to thank my family for their support and patience.

Chapter 1: Introduction

1.1 AIM OF THE THESIS

The daily process of searching for biomedical documents, such as scientific journal articles, in bibliographic databases is a common practice among professionals in the biomedical field. Given the criticality of this sector, it is important for searches to be fast and precise. However, a lot of times users search using terms that may not be sufficient for accurate retrieval. In this thesis, we will briefly attempt to find solutions for this problem.

We will examine the use of word embeddings and knowledge graphs for information retrieval in order to utilize their benefits in a query expansion scenario, by finding synonym terms to the terms of the query according to a simple similarity metric on these representations. We will also examine a reranking system that operates directly on word embeddings.

More specifically, we will use the popular technique word2vec (Micolov et al., 2013) that is used to represent words as dense vectors in a continuous vector space. Word2vec predicts the context in which a word appears, resulting in vector representations that capture semantic similarities and relationships between words. These embeddings enable us to explore semantic associations between terms, allowing for more comprehensive and contextually relevant query expansions in biomedical information retrieval tasks. We will also use knowledge graphs to refine our searches by leveraging relational structures and nodes. By encapsulating entities, concepts, and their interconnections within a structured framework, knowledge graphs offer a holistic view of domain-specific knowledge like that of the biomedical domain.

Furthermore, the retrieval system will be enhanced by a re-ranking system to ensure a better ranking of relevant documents in the results list. We will integrate transformers (Vaswani A. et al, 2017) by generating embeddings for each document and query, representing the contextual meaning of their text. Then we will compare the embeddings of the documents with the embedding of the query using the cosine similarity function.

We will evaluate these models on data from Round 5 of the TREC-COVID challenge. Preliminary results show that the proposed models outperform traditional ad-hoc retrieval using BM25.

In the course of this thesis, advanced models representing the modern trend in the literature were studied and implemented. The emphasis was on using dense word embeddings and document embeddings, knowledge graphs, as well as Transformers.

1.2 OUTLINE

Chapter 1

Chapter 1, named “Introduction”, outlines the aim of the study.

Chapter 2

Chapter 2, named “Methods”, provides a comprehensive overview of the methodologies employed in this thesis to enhance information retrieval systems.

Chapter 3

Chapter 3, named “Experiments and Results”, reports the experiments conducted during the work of this thesis, along with their results.

Chapter 4

Chapter 4, named “Related Work” reports recent work related to information retrieval methods.

Chapter 5

Chapter 5, named “Conclusions and Further work”, finalizes the experiment and suggests ideas for future work.

Chapter 2: Methods

This chapter describes the methods used in our experiments. We start by describing the BM25, a widely used ranking function that will be the baseline method for the rest of our methods. We then present two methods that try to improve the results of the BM25 by expanding the queries of the dataset with synonym words. The first one uses word2vec to find the synonym terms and the second uses knowledge graphs where we leverage relational structures and nodes for enhanced query expansion and refined searches. The last method we present is Re-ranking with Transformers.

2.1 AD-HOC RETRIEVAL

In this method, our goal is to develop a system to address the ad hoc retrieval task. This is the most standard 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. A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need (Manning, 2009). To estimate the relevance of documents to a given search query, the system will use the BM25 similarity function.

The BM25 Similarity Function

The BM25 (Robertson et al, 1995) is one of the most widely used ranking functions by search engines in an ad-hoc retrieval task. It is based on the probabilistic retrieval framework developed in the 1970s by Stephen E. Robertson, Karen Spärck Jones, and others. The function ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. Given a query *q* containing *n* terms q_1, q_2, q_n and a document *d*, BM25 is defined as follows:

$$\boldsymbol{score(q,d) = \sum_{i=1}^{n} idf(q_i) * \frac{tf(q_i,d) * (k_1\ + 1)}{tf(q_i,d) + k_1 * (1 - b + b * \frac{|d|}{avgdl}}}$$

Where:

$tf\ (q_i, d)$ is term's q_i frequency in document d_i.

$|d|$ is the length of the document d in words (terms) defined by: $|d\ | = 1/(norm * norm)$, where norm is the score factor used by Lucene's default similarity function.

$avgdl$ is the average document length over all the documents of the collection

k1 and *b* are free parameters, usually chosen as k1 = 2.0 and b = 0.75.a

$idf(q_i)$ is the inverse document frequency weight of the query term q_i

2.2 QUERY EXPANSION WITH SYNONYM WORDS

In this section, we introduce two methods that extend the BM25 model by making the retrieval process aware of synonyms. *Synonyms* are words that differ in spelling and pronunciation, but have the same or a very close meaning. For example, "pretty" and "gorgeous" are both synonyms of the word "beautiful". In information retrieval, it's common to use synonyms to decorate text in order to increase the probability that an appropriate query will match. The overall idea of synonym expansion is that when the search engine receives a stream of terms, it can enrich them by adding their synonyms, if they exist, at the same position. This enables documents, which do not necessarily contain the exact keywords entered by the user but which do contain other terms which carry a very similar meaning, to match the query.

The methods we use rely on the use of dense vectors and on materializing a semantic knowledge graph to find the synonyms of the query terms. The methods are presented below.

2.2.1 WORD2VEC-based Query Expansion

In this method, each query of the TREC-COVID dataset is first processed by a *text analysis pipeline*. A *synonym* filter in the pipeline uses a neural network to generate synonyms. The generated synonyms are then used together with the terms from the query to find matches in the inverted index using the BM25 function. Finally, the search results are collected.

In more details, we have a simple text analysis pipeline composed of a *tokenizer*, which creates a token every time it encounters whitespace, resulting in creating a token for each of the terms in a query. Then we use a *token filter* for synonym expansion: for each received token, it asks the neural network for synonyms and sees if any of the keywords is equal to the token text.

We use the *word2vec* neural network to learn a representation of the words that can tell us the most similar (or nearest neighbour) word for the query terms, and the *cosine distance* as a distance measure in order to find the most similar words with respect to a given word. Thus, we use these techniques to find a word’s synonyms. We briefly present them below.

Word2vec Algorithm

Word2vec is a popular technique that is used to represent words as dense vectors in a continuous vector space. The aim of Word2Vec models is to capture the semantic meaning of words based on their contextual usage in a given dataset-corpus. In Word2Vec, words that are similar in meaning are represented by vectors that are close to each other in the vector space. This is achieved by training a neural network on a large corpus of text data. The model learns to predict the context of a word within a sentence, and as a result, it assigns vector representations to words that capture their semantic relationships.

In (Mikolov et al, 2013) two different neural network models for learning such word representations are described: *continuous-bag-of-words* (CBOW) and *skip-gram*. Word2vec performs unsupervised learning of word representations; the mentioned CBOW and skip-gram models just need to be fed a sufficiently large text, properly encoded. The main concept behind word2vec is that the neural network is given a piece of text, which is split into fragments of a certain size (also called *windows*). Every fragment is fed to the network as a pair consisting of a *target word* and a *context*. The hidden layer of the network contains a set of weights (the number of neurons in the hidden layer) for each word. These vectors will be used as the word representations when learning ends. An important note about word2vec is that we don’t care much about the outputs of the neural network. Instead, we extract the internal state of the hidden layer at the end of the training phase, which yields exactly one vector representations for each word. During training, a portion of each fragment is used as target word, and the rest is used as context. With the CBOW model, the target word is used as the output of the network, and the remaining words of the text fragment (the context) are used as inputs. The opposite is true with the skip-gram model: the target word is used as input and the context words as outputs (as in the example). In practice, they both work well, but skip-gram is usually preferred because it works slightly better with infrequently used words. Skip-gram works well with small amount of training data and represents well rare words or phrases. On the other hand, CBOW is faster to train than skip-gram and has slightly better accuracy with frequent words.

After the model is trained, the learned word embeddings are positioned in the vector space such that words that share common contexts in the corpus — that is, words that are semantically and syntactically similar — are located close to one another in the space. More dissimilar words are located farther from one another in the space. Using a similarity measure like cosine similarity function, we can take advantage of this and find the most similar words to use as synonym words.

Cosine Similarity Function

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between the two vectors and determines whether the vectors are pointing in roughly the same direction. Given vectors $\vec{x}$, $\vec{y}$ we define

$$similarity(\vec{x},\vec{y}) = \cos(\theta) = \frac{\vec{x}\ \cdot \vec{y}}{||\vec{x}||\,||\vec{y}||}$$

Where:

θ is the angle between the vectors.

$\vec{x}\ \cdot \vec{y}$ is the dot product between $\vec{x}$ and $\vec{y}$ and is calculated as:

$$\vec{x} \cdot \vec{y} = \vec{x}^T \vec{y} = \sum_{i=1}^{n} \vec{x}_i \vec{y}_i = \vec{x}_1 \vec{y}_1 + \vec{x}_2 \vec{y}_2 + \cdots + \vec{x}_n \vec{y}_n$$

$||\vec{x}||$ represents the L2 norm or magnitude of a vector $\vec{x}$ which is calculated as

$$||\vec{x}|| = \sqrt{\vec{x}_1^2 + \vec{x}_1^2 + \cdots \vec{x}_1^n}.$$

The similarity can take values between -1 and +1. Smaller angles between vectors produce larger cosine values, indicating greater cosine similarity. In our case, the vectors are the outputs of the word2vec neural network, i.e. the representations of the words of the dataset.

2.2.2 Semantic Knowledge Graph-Based Query Expansion

In this method, we built a Semantic Knowledge Graph (SKG) from TREC-COVID corpus. An SKG is able to dynamically discover and score interesting relationships between any arbitrary combination of entities (words, phrases, or extracted concepts) through dynamically materializing nodes and edges from a compact graphical representation built automatically from the corpus. In this vein, we built an SKG and use it to discover synonym words for the terms of each query. The discovered synonyms are then used together with the terms from the query to find matches in the inverted index using the BM25 function. Finally, the search results are collected.

A common search engine usually retrieves documents relevant to a specific query. We can imagine an SKG as a search engine that instead finds and ranks terms that best match a query. Next, we present the structure of an SKG and give an insight on how it discovers and scores semantic relationships.

Semantic Knowledge Graphs

In this section, we describe a new kind of knowledge representation and mining system which is called the Semantic Knowledge Graph (Grainger et al, 2016). The Semantic Knowledge Graph leverages an inverted index, along with a complementary uninverted index, to represent nodes (terms) and edges (the documents within intersecting postings lists for multiple terms/nodes). This provides a layer of indirection between each pair of nodes and their corresponding edge, enabling edges to materialize dynamically from underlying corpus statistics. As a result, any combination of nodes can have edges to any other nodes materialize and be scored to reveal latent relationships between the nodes.

Structure

A Semantic Knowledge Graph is organized as a set of nodes and edges, where nodes represent entities and edges represent relationships between those entities. Entities can be anything from concepts, objects, or entities in the real world. Each node corresponds to a specific entity. The relationships between entities, represented by the edges, are semantically meaningful and denote connections, associations, or dependencies between the entities. Edges may have associated weights or scores that quantify the strength or significance of the relationship between the connected entities. In our case, the entities are the vocabulary terms of the dataset, and the edges are sets of documents (this notion will be analysed later). This graph structure highlights the connectivity between nodes, illustrating how different entities are related to each other through direct or indirect relationships.

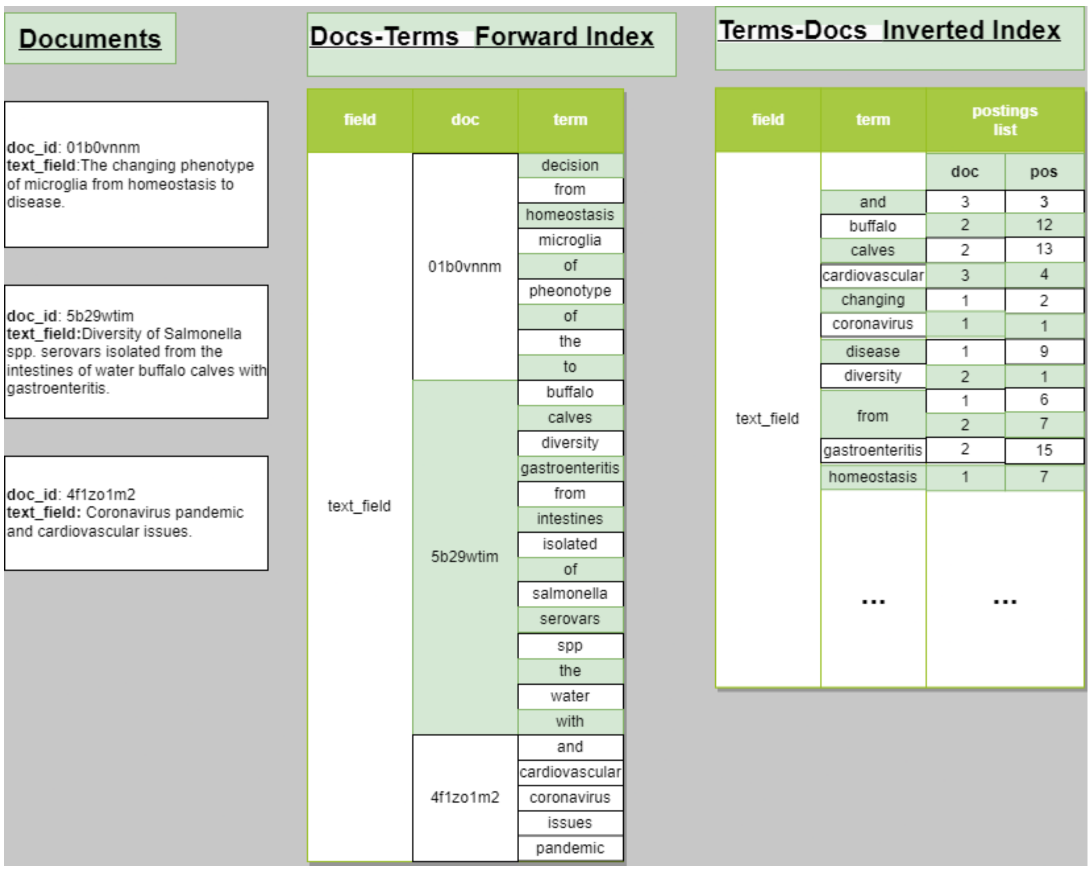
Discovering and Scoring Semantic Relationships

In order to *materialize* an SKG we created an inverted index and a corresponding forward (uninverted) index from the documents of TREC-COVID Complete. These indexes serve as the underlying data structure which enables real-time traversal and ranking of any arbitrary semantic relationships present within our collection of documents. The inverted index is initially used to identify documents that contain a specific query term. It serves as the starting point for identifying potential semantic relationships between terms. By finding documents that contain the original query term, the inverted index enables the retrieval of co-occurring terms within those documents. The forward index maps documents to the terms they contain. It provides a reverse lookup mechanism compared to the inverted index, allowing for retrieval of terms associated with specific documents. Figure 3.1 demonstrates how documents get mapped into both the forward index and the inverted index.

On the left of the figure, there are three *demo* documents from TREC-COVID, each of which has a *doc_id* and *text_field*. The right side of the figure shows how these documents are mapped into the search engine. The inverted index maps the *text_field* to a list of terms, and then maps each term to a postings list containing a list of documents (along with positions in the documents, as well as some other data not included in the figure). This makes it quick and efficient to look up any term in *text_field* and find the set of all documents containing that term. The forward index for the *text_field* maps each document to a list of terms contained within that document.

The synonym words to a query term are found by first executing a *search* for the term on the inverted index. The index returns a set of documents that contain this term. Then we *look up* these documents at the forward index, which returns their terms. In essence, by doing two traversals (terms to documents to terms) we find all of the terms that appear together in documents containing the original query term.

The next step, is to *filter* these terms and select the ones considered to be the most semantically related to the query term. This is accomplished by applying a *relatedness function*. The concept of relatedness functions revolves around discerning semantic connections between terms within a set of documents. Specifically, given some documents containing a first term *x*, any second term *y* tends to be semantically similar to *x* if it co-occurs in the same documents as *x* more often than it co-occurs in documents with other random terms. Of course, this is based on the Distributional Hypothesis that suggests "the more often two terms appear together within the documents the more similar they will be". The terms with the greater score are considered to be synonym words and are used to expand the query.



Documents

doc_id: 01b0vnnm
text_field:The changing phenotype of microglia from homeostasis to disease.

doc_id: 5b29wtim
text_field: Diversity of Salmonella spp. serovars isolated from the intestines of water buffalo calves with gastroenteritis.

doc_id: 4f1zo1m2
text_field: Coronavirus pandemic and cardiovascular issues.

Docs-Terms Forward Index

field	doc	term
text_field	01b0vnnm	decision
		from
		homeostasis
		microglia
		of
		pheonotype
		of
		the
		to
	5b29wtim	buffalo
		calves
		diversity
		gastroenteritis
		from
		intestines
		isolated
		of
		salmonella
		serovars
		spp
		the
		water
		with
	4f1zo1m2	and
		cardiovascular
		coronavirus
		issues
		pandemic

Terms-Docs Inverted Index

field	term	postings list	
		doc	pos
text_field	and	3	3
	buffalo	2	12
	calves	2	13
	cardiovascular	3	4
	changing	1	2
	coronavirus	1	1
	disease	1	9
	diversity	2	1
	from	1	6
		2	7
	gastroenteritis	2	15
	homeostasis	1	7
	...	...	

Figure 3.1 Forward and Inverted Indexes

The method is further distilled with the application of *weights* on the terms of the query. A number of possible query permutations is explored when rewriting the queries in order to include enhanced semantic context, and consider a trade-off between recall and precision.

2.3 RE-RANKING USING TRANSFORMERS

In this method, we extend our search methodology by integrating transformers and applying document re-ranking. To integrate transformers, we first generate embeddings for each document and query, representing the contextual meaning of their text. Several pre-trained transformers models are used in this step. The document embeddings along with their text fields are stored in an index. The BM25 function is used on the index for ad-hoc retrieval. The documents retrieved for each query are re-ranked as follows: we compare the embeddings of the documents with the embedding of the query using the cosine similarity function. This comparison assigns a similarity score between each document and the query. This score is combined with the score from BM25. The final similarity score between a query and a document is the sum of the BM25 score from the ad-hoc retrieval and the cosine similarity score of the embeddings. A weighting scheme is also applied to the scores of the two methods, with the weights determined through testing. Finally, the documents are sorted according to this score from highest to lowest.

We briefly present the method as well as transformers below.

Transformers

A transformer (Vaswani A. et al, 2017) is a neural network architecture, alternative to traditional recurrent and convolutional models, specifically tailored for sequence-to-sequence tasks in natural language processing. A transformer uses the attention mechanism to weight the importance of different parts of the input sequence when making predictions allowing them to capture long-range dependencies and relationships. Unlike traditional recurrent neural network (RNN) or long short-term memory (LSTM), where information is passed sequentially from one step to the next, in transformers, self-attention allows the model to consider all positions in the input sequence simultaneously, capturing long-range dependencies more effectively. Also, multiple attention heads are used to capture different aspects and patterns in the data. Each head learns a different set of attention weights. The outputs from multiple heads are concatenated and linearly transformed to generate the final output. The architecture includes an encoder to process the input sequence and a decoder to generate the output sequence. Each encoder and decoder layer in a transformer contains a feedforward neural network, providing non-linear transformations to the input data.

For information retrieval tasks, transformers can be used to represent and understand the content of documents and queries.

Generating Documents Embeddings

To generate document embeddings, we have to either train a model from scratch or use one of the many pre-trained models openly available on the internet. While Transformers enable state of the art language models to be built, having the knowledge and resources to build them from scratch can present a large hurdle. In order to avoid the large amount of expensive computing power and time needed to train a model from scratch we have chosen to apply a pre-trained model as a starting point.

In practice, it is easy to get the embedding of a document using a pre-trained model, given the capabilities of Sentence Transformers[1] and S-BERT. S-BERT (Sentence-BERT), modifies the pre-trained BERT network to use siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be computed using cosine similarity (Reimers and Gurevych, 2019). It is a Python framework provided by Hugging Face[2]. Hugging Face provides many different models that have been trained on plenty of datasets. For our purposes we did not use a specialized model for our dataset. We preferred a more generic one that has been trained on various and large datasets. Each model has its own dimensional dense vector space depending on the scale of the model.

[1] https://www.sbert.net/
[2] https://huggingface.co/

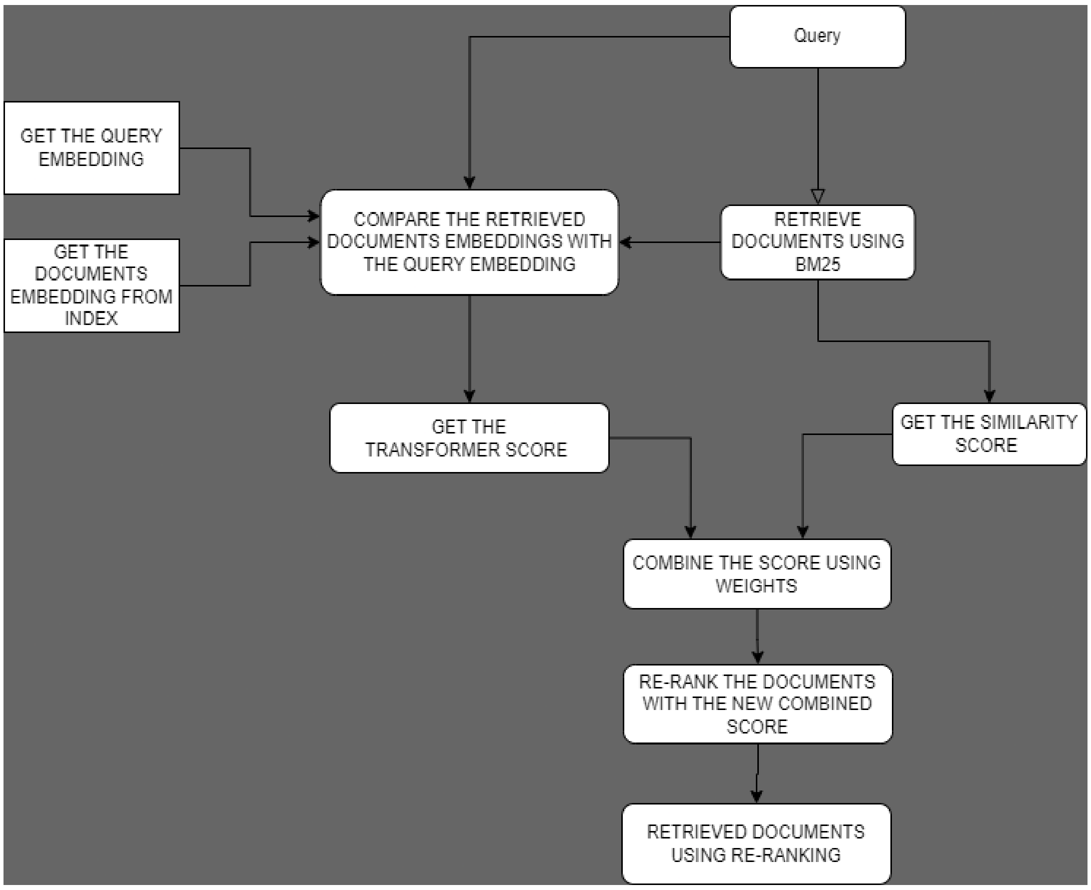

Query
GET THE QUERY EMBEDDING
GET THE DOCUMENTS EMBEDDING FROM INDEX
COMPARE THE RETRIEVED DOCUMENTS EMBEDDINGS WITH THE QUERY EMBEDDING
RETRIEVE DOCUMENTS USING BM25
GET THE TRANSFORMER SCORE
GET THE SIMILARITY SCORE
COMBINE THE SCORE USING WEIGHTS
RE-RANK THE DOCUMENTS WITH THE NEW COMBINED SCORE
RETRIEVED DOCUMENTS USING RE-RANKING

Figure 3.2 *Re-Ranking Explained*

Re-ranking method

The re-ranking process aims to refine the initially retrieved documents, optimizing their order and relevance. The goal is to enhance the precision of the search results and provide users with more accurate and relevant documents in response to their queries. Figure 3.2 explains the process of re-ranking.

Chapter 3: Experiments and Results

3.1 DATA

The evaluation of the methods described in this work utilize data provided by the TREC-COVID Challenge ().

3.1.1 TREC-COVID Complete corpus

TREC-COVID challenge

The TREC-COVID challenge[3] was a collaborative effort involving the Allen Institute for Artificial Intelligence (AI2), the National Institute of Standards and Technology (NIST), the National Library of Medicine (NLM), Oregon Health & Science University (OHSU), and the University of Texas Health Science Center at Houston (UTHealth). Utilizing the TREC model, the challenge developed a series of Information Retrieval (IR) test collections based on the CORD-19 datasets (see (Wang et al, 2020) for more details on CORD-19).

The primary objectives of the challenge were to assess search algorithms and systems that could aid scientists, clinicians, policy makers, and others in navigating the continually expanding body of scientific literature related to COVID-19. Additionally, the aim was to identify methods that could be instrumental in managing scientific information during future global biomedical crises.

The challenge unfolded in five distinct rounds, each of which is now concluded. For every round, organizers specified a particular version of the CORD-19 dataset to be employed, along with releasing a set of information need statements known as topics. Following the submission deadline for a round, NIST utilized the received runs to generate, for each topic, a collection of documents evaluated for relevance by human annotators. Annotators were selected from NLM, OHSU, and UTHealth to ensure they possessed the necessary biomedical expertise. The relevance assessments were employed to calculate effectiveness scores for the retrieval runs.

[3] https://ir.nist.gov/trec-covid/

All the topics, runs, scores, and relevance judgments are archived on the TREC-COVID website, accessible to everyone free of charge. The final cumulative collection, TREC-COVID Complete, is now available as a standard ad hoc retrieval test collection and serves as the dataset for the experiments conducted in this study.

Dataset description

The dataset is curated from the CORD19 corpus (Wang et al., 2020). CORD-19[4] is a corpus of academic papers about COVID-19 and related coronavirus research. The Round 5 of TREC-COVID uses the July's 16, 2020 release of CORD19, which includes 140.000 articles and a set of 50 topics related to COVID-19.

Topics

The topics used in TREC-COVID have been written by the organizers with biomedical training with inspiration drawn from consumer questions submitted to NLM, discussions by "medical influencers" on social media, and suggestions solicited on Twitter via the #COVIDSearch tag. They are representative of the high-level concerns related to the pandemic.

The Round 5 topic set contains 50 topics. Each topic consists of three fields. A *query* field, a *question* field, and a *narrative* field. The *query* field provides a short keyword statement of the information need, while the *question* field provides a more complete description of the information need. The *narrative* provides extra clarification, and is not necessarily a super-set of the information in the question. In our experiments we use the *query* field. The use of the other fields has no impact to the methods examined.

Pre-processing

In this section we are going to analyze all the process we follow before making any searches. First, we have to read-load our documents from the corpus. We only keep the abstract from the documents, which contain the *title, abstract description, doc id, url, pubmed id.*

After we have read our documents, we need to analyze the text. First, we need to break down raw text into terms or tokens that can be efficiently indexed and retrieved. By doing that we create a structured and normalized representation of the text that allows

[4] CORD19 can be downloaded from https://github.com/allenai/cord19

for a more accurate and relevant retrieval of information. The steps that we will follow are Tokenization, stop words removal and Stemming

Tokenization breaks down the text into individual tokens or terms, furthermore it splits the text into smaller units, such as words or phrases. This process makes it easier to create an index of terms, which is fundamental for efficient search operations. As we progress in our analysis, we understand how valuable this process is and also simple at the same time.

Following tokenization now we can process our terms and remove the stop words. Removing stop words helps reduce the noise of our index and improves search efficiency by focusing on more meaningful terms. Each document may contain many stop words more than once, stop words do not provide any value when searching for them as they are present in every document. The absence of stop words allows the search engine to focus on more meaningful and specific terms, this leads to more accurate and relevant search results.

Stemming is a popular text normalization technique that is used in information retrieval. What stemming does is that it reduces words to their base or root form. The purpose we are using stemming is that it groups variations of words that have the same root. Recognizing, searching and retrieving more forms of words returns more results. When a form of a word is recognized, it's possible to return search results that otherwise might have been missed. That additional information retrieved is why stemming is integral to search queries and information retrieval.

Indexing

For storing all of our data into an index we have used bulking methods that are provided by Elastic Search. Since the dataset is quite big (around 170.000 documents), if we decided to pass one by one all the documents and store all the necessary fields, it would take too much time. For that issue we efficiently use the bulk API to index multiple documents in a single request instead of having one request for each document. For each document we have indexed its *doc id. Title. Published date and text (abstract).* Furthermore, we have created a field that contains both the Title and the Text of each document, by doing that we can search in this field without losing any context of the whole document.

3.2 TOOLS

In this section, the basic tools used in this work are briefly presented. For the experiments we have used a wealth of libraries that made our progression faster and simpler.

ElasticSearch[5]
Elasticsearch is a distributed search and analytics engine built on Apache Lucene. ElasticSearch offers simple REST-based APIs, a simple HTTP interface, and uses schema-free JSON documents, making it easy to get started and quickly build applications for various use cases. The distributed nature of Elasticsearch enables it to process large volumes of data in parallel, quickly finding the best matches for our queries. In abstract Elasticsearch helps us with fast time-to-value, has high performance abilities and near real-time operations. Elasticsearch comes integrated with Kibana, a popular visualization and reporting tool.

Kibana[6]
Kibana is a visualization and exploration tool that complements Elasticsearch. It provides a user-friendly interface for interacting with Elasticsearch data, allowing users to create diverse visualizations such as charts, graphs, and maps. Kibana enables real-time analysis and monitoring of data stored in Elasticsearch, making it a crucial component in the Elastic Stack. Users can perform searches, analyze trends, and gain insights into their data through customizable dashboards, making it an essential tool for anyone leveraging Elasticsearch for data storage and retrieval.

For the installation of Elasticsearch and Kibana we have used Docker.

Docker[7]
Docker is a software platform that allows us to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that include everything the software needs to run including libraries, system tools, code, and runtime. Docker enables consistent application deployment across different environments, simplifying development, testing, and deployment processes. The technology promotes isolation, portability, and scalability.

---

[5] https://www.elastic.co/elasticsearch
[6] https://www.elastic.co/kibana
[7] https://www.docker.com/

We are relying on Docker for the installation and orchestration of Elasticsearch and Kibana. The abstraction provided by Docker facilitates a consistent and reproducible setup, making more efficient the process of integrating these tools into our project.

For the experiments we have used the following versions:

Elasticsearch = 7.17.13

Kibana = 7.17.13

Gensim[8]

We have used Gensim (Python Library) for converting our documents and queries to word embedding. Word2Vec in Gensim, transforms words into numerical vectors, capturing semantic relationships between them. It learns to represent words in a continuous vector space, enabling similarity comparisons and semantic relationships. Gensim simplifies the process of training and using Word2Vec models, making it accessible for tasks like natural language processing and machine learning applications.

Trec_Eval[9]

Trec_Eval is an evaluation software which is used to evaluate rankings, either documents or any other information that is sorted by relevance. The evaluation is based on two files: The first, known as "qrels" (query relevance), contains the relevance judgements for each query. The second contains the rankings of documents returned by our Information Retrieval system. We are using trec_eval for measuring and evaluating our system.

3.3 EVALUATION MEASURES

Before delving into the analysis of the developed models and the corresponding results, it is crucial to establish the metrics that will serve as the basis for assessing the performance throughout the remaining of this thesis.

3.3.1 TF-IDF

TF stands for term frequency whereas IDF stands for inverse document frequency. TF-IDF is one of the most common used metric for information retrieval. Even though it is simple it leads to very good results. TF-IDF, as a numerical statistic, assesses the

[8] https://radimrehurek.com/gensim/
[9] https://github.com/usnistgov/trec_eval

significance of words in a given text by assigning scores, taking into account the corpus to which the document belongs. Several studies have explored this method in the context of extracting keywords and determining the relevance of words. The bigger the score, the more significant the word is.

This how the TF is being calculated:

$$\mathrm{TF} = \frac{\mathrm{count(w)}}{N}$$

Where:

N is the total number of words (tokens) that are present in the document

$count(w)$ is the total number of word occurrences in the document

$$IDF(w) = \log(\frac{N}{M(w)})$$

Where:

N is the total number of documents inside the dataset

$M(w)$ is the total number of documents that contain at least once the word w.

$$TF\_IDF(w) = TF(w) * IDF(w)$$

3.3.2 Common Evaluation Measures

Evaluation measures for an Information Retrieval system assess how well an index, search engine or database returns results from a collection of resources that satisfy a user's query. The performance of an IR system can be evaluated based on various factors such as speed, user contentment, efficiency. Nevertheless, the standard approach to information retrieval system evaluation revolves around the notion of *relevant* and *nonrelevant* documents. With respect to a user information need, a document in the test collection is given a binary classification as either relevant or nonrelevant. This decision is referred to as the *gold standard* or *ground truth* judgment of relevance.

Given these ingredients, the two most frequent and basic measures for information retrieval effectiveness are precision and recall.

Precision measures the ability of a system to retrieve only relevant items, indicating the proportion of retrieved documents that are relevant.

$$Precision = \frac{TP}{TP + FP}$$

Recall measures the ability of a system to retrieve all relevant documents.

$$Recall = \frac{TP}{TP + FP}$$

Where:

TP (True Positives): The number of relevant documents that were correctly retrieved.

FP (False Positives): The number of irrelevant documents that were incorrectly retrieved.

FN (False Negatives): The number of relevant documents that were incorrectly not retrieved.

There is a trade-off between retrieving all relevant documents (high recall) and ensuring that the retrieved documents are indeed relevant (high precision). So, in general, higher recall means lower precision and higher precision means lower recall. The measures are first defined for the simple case where an IR system returns a set of documents for a query and do not take into consideration the order of the documents that are being retrieved. We will see next how to extend these notions to ranked retrieval situations.

Precision at k($\boldsymbol{P@k}$)

Precision at k is a metric that measures the precision of our system at a specific cutoff point, denoted as k. Precision at k is calculated as the ratio of relevant documents among the top k retrieved items.

$$P@k = \frac{TP\ among\ the\ top\ k\ retrieved\ documents}{k}$$

Recall at k($\boldsymbol{R@k}$)

Recall at k is a metric that measures the recall of our system at a specific cut-off point_k. Recall at k is calculated as the ratio of relevant documents that have been retrieved to the total number of relevant documents in the dataset.

$$R@k = \frac{TP\ among\ the\ top\ k\ retrieved\ documents}{Total\ number\ of\ relevant\ documents}$$

Mean Average Precision (MAP)

MAP is another very common metric that is being used to evaluate the performance of information retrieval systems, particularly in the context of ranked retrieval results. MAP represents the average precision score for various levels of recall. In order to better understand MAP we need first to explain what Average Precision (AP) is.

AP considers the precision at each relevant document in the ranked list

$$AP = \frac{\sum_{k=1}^{n} P@k \ x \ Rel_k}{Total\, number\ of\ relevant\ Documents}$$

Where:

$P@k$ Stands for the Precision at a given cutoff point k and is the ratio of relevant documents among the top k retrieved documents.

Rel_k is an indicator function equal to 1 if the document at rank k is relevant and 0 if not

Now that we understand what AP is, we can better understand what MAP does. MAP takes the Average Precision values for multiple queries and calculates the average (mean) precision across all the queries. In other words, it provides a measure of the average effectiveness of a retrieval system across a set of queries. MAP is one of the most useful tools we have to evaluate the overall performance of our information retrieval system.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Where:

N Is the total number of queries.

In abstract:

Average Precision (AP): Computes the Average Precision for each query independently, as explained above

Mean Average Precision (MAP): Average the AP values across all queries

Normalized Discounted Cumulative Gain$(\boldsymbol{NDCG})$

NDCG is a metric that is being used to evaluate the ranking quality in the set of search results. The theory is that the most relevant documents should be ranked better than the irrelevant documents. With relevant we mean the score that each document has that indicates how relevant the document is in a query. The higher the score the more relevant the document is. This relevancy is obtained either by supervised or semi-supervised ways, in our case it is supervised. [https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0]

To better understand the *NDCG* measure, we need to break it down into its constituent parts.

Cumulative Gain ($\boldsymbol{CG}$)

CG is the sum of the score that is associated with the documents that are being retrieved in a query.

$$CG = \sum_{i=1}^{N} Document\_Score_i$$

For better understanding we will have an example.

Let's say we have these two systems that both retrieved 4 documents for the same query. The document score for each system is shown below:

Documents / System	D1	D2	D3	D4
X	0	1	2	1
Y	2	0	0	2

$$CGx = 0 + 1 + 2 + 1 = 4$$

$$CGy = 2 + 0 + 0 + 2 = 4$$

The score represents how relevant the document is, higher the score the more relevant the document is. As we can see the CG score is the same for both systems and we do not know which one is better for that reason we implement the Discounted Cumulative Gain (DCG)

Reminder: In our scenario we do not have score higher than 1. A document is either relevant or not but for the sake of the example we use scores.

Discounted Cumulative Gain ($\boldsymbol{DCG}$)

DCG is almost the same as CG but takes into consideration that the highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result. With the above metric we can solve the problem that we faced with Cumulative Gain. [https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0]

$$DCG = \sum_{i=1}^{N} \frac{Document\_score_i}{log_2(i+1)}$$

So, the score for the example above will be:

$$DCG_x = \frac{0}{log_2(1+1)} + \frac{1}{log_2(2+1)} + \frac{2}{log_2(3+1)} + \frac{1}{log_2(4+1)} \approx 2.0616$$

$$DCG_y = \frac{2}{log_2\text{x}(1+1)} + \frac{0}{log_2(2+1)} + \frac{0}{\text{log}log_2(3+1)} + \frac{2}{log_2(4+1)} \approx 1.4307$$

Now we know that the system x is better than the system y according to DCG. It makes sense as the documents in the higher rank are more relevant (higher score). However, we prefer to use nDCG instead of DCG and the reason behind that is that is normalized.

Normalized Discounted Cumulative Gain($nDCG$) :

$nDCG$ Provides a score between 0 and 1, where 1 indicated perfect ranking and 0 indicates the worst possible ranking. This consistent scale allows for easier interpretation and comparison of the effectiveness of the different systems.

$$nDCG = \frac{DCG}{Ideal\ DCG}$$

More about how $iDCG$ is calculated can be found here:

https://en.wikipedia.org/wiki/Discounted_cumulative_gain

https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0

3.3.3 TREC-COVID Evaluation Measures

For the TREC-COVID evaluation we adhere to the official evaluation measures of the challenge. The challenge requires a maximum of 1000 documents to be returned for each query. According to the challenge's guidelines, the measures used in the 5th phase of the competition are P@20, NDCG@20, and MAP. We also record P@5, P@10, Recall and NDCG@10

MAP: The standard definition of Mean Average Precision calculated on the 1000 top documents retrieved.

Precision@20: Precision calculated on the top 20 documents retrieved.

nDCG@20: The normalized discounted cumulative gain calculated on the top 20 documents retrieved.

3.4 EXPERIMENTAL SETUP

3.4.1 First-Stage Retrieval

Initially, we index the document collection using the *ElasticSearch* search engine. We employed standard pre-processing methods, encompassing English *stopword removal* and *Porter stemming*. We want to note that it is crucial to pre-process both the documents and queries using the same analyzer. We used the *EnglishAnalyzer* of *ElasticSearch*. The text utilized for indexing is a concatenation of the title and abstract. For the topics we only keep the query field (nothing from question or narrative fields). This is done in previous works. Each query contains only a few terms, no more than 6 or 7. Subsequently, we execute the queries on the index for each topic using the BM25 scoring function with default parameters (k1 = 0:9, b = 0:4). Within this system, a fixed threshold of 1000 is applied, ensuring only the top 1000 BM25 results are retrieved. This retrieval setup will be referred as BM25-BASE.

3.4.2 Word2vec-based Query Expansion

Prior to training the word embeddings, both documents and queries were analyzed using the EnglishAnalyzer of ElasticSearch. For the training of the word embeddings, we used the word2vec implementation of the *gensim* Python toolkit[10] using the CBOW model with a window of 6 words before and 6 words after the central word, negative sampling, 30 epochs, and embedding dimensions r = 400. Also, minimum occurrences of terms were set to 10. All other parameters were set to the default values of *gensim*.

[10] https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

To measure the similarity between the word embeddings and find synonym words, we utilized the cosine similarity function as described in section 3.2.1.

In more details, we used the word2vec output results as synonyms only if they have a similarity greater than a certain threshold. A filter picks only the two words closest to the given query term (according to word2vec) having a similarity of at least 0.68, for each query term passed by the tokenizer.

As an additional step, we take into consideration the similarity score and assign it as a weight to each synonym word, i.e. we multiply the word with the corresponding weight. Higher similarity score results in higher weight for a word. Weights serve as a mechanism to indicate the importance or significance of each word, and influence how much impact a particular term has on the overall relevance of a document in the search results. For example, given the term "mask" we find the most similar word to be "facemask" and expand the query with the "facemask" multiplied by the weight of the cosine similarity score between them. The original terms of the query are multiplied by 1.0.

We further refined this step by adjusting the assigned weights according to the following observation. We observed that the term "coronavirus" and/or its synonym terms (sars2, covid, etc.) appear in almost every query. These terms also appear in the vast majority of the dataset's documents. Thereby, there is little discriminative value that can be granted to them. Given that, we created a list of all words similar to the "coronavirus" term, which we call "*covid-list*". This list is being used in order to adjust the weights of the original query terms and its synonyms. If an original term is not included in this list its weight is increased by 20%, otherwise it is decreased similarly. If a synonym is included in this list its weight remains the same, otherwise it is decreased in half.

In Table 4.1 we can see the transformations of a query.

Original query: impact of masks on coronavirus transmission (words have received stemming)		
Query term	**Synonym word**	**Similarity weight**
impact	consequ	0.6271
mask	facemask	0.7980
transmiss	spread	0.6195
Final query: impact 1.0 consequ 0.6271 mask 1.0 facemask 0.7980 coronaviru 1.0 transmiss 1.0 spread 0.6195		
Final query with adjusted weights: impact 1.2 consequ 0.6271 mask 1.2 facemask 0.7980 coronaviru 0.85 transmiss 1.2 spread 0.6195 sars 0.35 covid 0.33		

Table 4.1 Synonym expansion and assignment of weights to the terms and synonym terms of a query.

We will refer to the model that uses the weights with no adjustment as QE-W2VW-BM25, whereas the model that adjusts weights will be referred to as QE-W2VAW-BM25.

3.4.3 Semantic Knowledge Graph-Based Query Expansion

During the indexing process, Elasticsearch processes the documents, tokenizes the text, and builds both the inverted index and the forward index.

To materialize the SKG, we first identify documents containing the original query term using the inverted index. Then the forward index is accessed to retrieve the terms present in those documents. By traversing from documents back to terms using the forward index, the SKG construction process gathers additional semantic context surrounding the original query term.

To measure the semantic relatedness between words, we use Elasticsearch’s aggregations framework. Aggregations group and perform calculations and statistics (such as sums and averages) on the data by using a simple search query. An aggregation can be viewed as a working unit that builds analytical information across a set of documents. In our case, we employ the *significant_text* and *significant_terms* aggregations. *Significant_text* aggregation uses a combination of term frequency, statistical analysis, and semantic modeling to calculate the relevance score between words within a given context. This type of aggregation uses both the inverted and forward index for its traversals and identifies terms that are not only frequent within a specific set of documents but have also a meaningful semantic relevance. The process begins with a standard query, where for each term of the query, documents containing the specified term are retrieved. Then the *significant_terms* aggregation utilizes the *significant_text* aggregation results and analyzes the distribution of terms within the retrieved documents (forward index) in comparison to a background set (all documents in the index). The result of this aggregation is a list of terms deemed statistically significant in the context of the provided query term.

We enforce a minimum threshold score (edge score) for adding these terms to the query, allowing each query term to have zero to multiple related terms. We tested and found that terms with score from 15 to 20 are semantically related. After expanding the query, we employ BM25 similarity scoring for analysis and search, enhancing semantic retrieval compared to conventional BM25 searches.

Exploring different query expansion techniques

The expanded queries have many capabilities if we modify the minimum match thresholds and percentages, which can tilt the scale between recall and precision. We experiment on the use of weights for the terms of the queries. The weights of the terms are determined in relation to the edge score that is being calculated with the aggregation queries. Each term is multiplied with its weight. The threshold is lowered and set to 3, so more terms are added to each query. The "*covid_list*" is also used accordingly. We summarize the query permutations below:

Simple Query Expansion – No Weights

The simple query expansion is the same as previously described, matching any documents containing either the original query or any of the semantically-related synonym terms and no weights added.

Simple Query Expansion – Weights

This is the same as above with the extra feature that all terms (original + synonyms) are multiplied by their weight.

Increased Precision – Reduced Recall Query Expansion:

We set a minimum match of at least 30% between the terms of the query and the documents. So, in order for a document to be considered relevant it must contain at least the 30% of the terms of the query.

Increased Precision – No reduction in Recall Query Expansion:

We require the original query term to match and also require at least another term to match (original query term plus one). So, in order for a document to be retrieved it must contain the original query term plus one from the synonyms.

Increased Recall Query Expansion:

We require two synonym terms to match the documents, but we do not explicitly require the original query term to match. This means that documents which are conceptually similar but don't necessarily contain the original query term can be retrieved.

We will refer to the models that result from the query permutations (in the order described) as QE-SKG-SNW-BM25, QE-SKG-SW-BM25, QE-SKG-IPRR-BM25, QE-SKG-IPNR-BM25.

3.4.4 Re-Ranking using Transformers

Sentence Transformers Model Implementation

To generate S-BERT sentence embeddings with Hugging Face, we imported the *Autotokenizer* and *Automodel* to tokenize and create a model from a pre-trained S-BERT model. The pre-trained models used were *all-mpnet-base-v2*[11], *all-MiniLM-L6-v2*[12], and *ClinicalBERT*[13] with 384- or 768-dimensional dense vector space. Pytorch was used to compute the embeddings. A list was created that contained (document embedding, document id) pairs. We saved those pairs to a pickle[14] file, for easy access on later uses. Also, we updated the index making good use of ElasticSearch vectors capabilities. We defined a mapping for our index, specifying that we had a *vector* field and loaded the generated embedding of each document there. The same procedure was done for the queries. For the queries we used both *query* field and *narrative* field of

[11] https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[12] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[13] https://huggingface.co/medicalai/ClinicalBERT
[14] https://docs.python.org/3/library/pickle.html#module-pickle. The pickle module implements binary protocols for serializing and de-serializing a Python object structure.

TREC-COVID. Narrative contains more information about the question so we can create a more meaningful embedding that we can compare with the documents.

Re-Ranking details

We applied *vector search* using cosine similarity measure to the results of a first-stage retrieval with BM25 (default parameters). Each document is assigned a score which is a mix of both the BM25 method and the score from the vector search. Weights are also applied, in favor of the transformers' scores. Documents are re-ranked according to this combined score from highest score to lowest.

We will refer to the approach that uses *ClinicalBERT* with RR-Clinical, the approach that uses *all-MiniLM-L6-v2* model with RR-Mini and the approach that uses *all-mpnet-base-v2* with RR-Mega.

3.5 EXPERIMENTAL RESULTS

In this section, we present the results of the methods that we used in the experiments on TREC-Covid dataset.

3.5.1 Query expansion experiments

Table 3.1 reports the performance of the query expansion methods. We can see that the knowledge graph-based query expansion with weights (relevance scores of the terms) (QE-SKG-SW-BM25) leads to an increase in recall compared to the other models, which is critical in medical systems. The use of weights on the synonyms derived from the graph is better that the no-weighting scheme (QE-SKG-SNW-BM25). This is not true for the word2vec-based query expansion method where the use of weights to the scores of synonym terms has a negative effect compared to not using weights. The knowledge graph-based query expansion with weights outperforms the other methods in almost all cases, and has an almost similar score with BM25 in the cases that comes second.

Method / Metrics	BM25 BASE	QE-W2V-BM25 SKIP	QE-W2VW-BM25 SKIP	QE-W2V-BM25 CBOW	QE-W2VW-BM25 CBOW	QE-SKG-SNW-BM25	QE-SKG-SW-BM25
Relevant Retrieved	9955	9998	10001	10014	10150	10189	**10210**
R@1000	0.4121	0.416	0.4181	0.42	0.425	0.429	**0.432**
P@5	0.7080	0.672	0.624	0.67	0.63	0.6840	0.7040
P@10	0.6640	0.648	0.5954	0.62	0.58	0.66	**0.67**
P@20	**0.62**	0.582	0.57	0.59	0.56	0.618	0.6140
Ndcg@10	**0.6007**	0.57	0.5579	0.548	0.5	0.5763	0.5965
Ndcg@20	0.5790	0.54	0.5261	0.54	0.49	0.5527	**0.5964**
MAP	0.22	0.2064	0.201	0.21	0.208	0.22	**0.23**

Table 3.1: Evaluation metrics for word2vec-based and knowledge graph-based query expansion models with and without weights.

To get a sense of what is learned by each of the expansion models, in Table 3.2 we report the top synonym terms for two sample queries from the TREC-Covid dataset. According to this table, the terms added to the queries by the word2vec model are syntactically or semantically related to individual query terms, which is expected. For the query "covid phylogenetic analysis" as an example, the terms "phylogenomic", "phylogenies", and "phylogeographic" are related to the query term "phylogenetic", while the terms "ethnographic" and "analisi" are related to the query term "analysis". The term "analisi" is the Italian word for analysis, and has probably resulted from a group of Italian documents available in the corpus. In contrast, looking at the synonym terms obtained by the knowledge graph, we can see that some terms are relevant to the whole query, and describe its subject matter. In a sense it is like the answer to the query is constructed. For example, a "phylogenetic analysis" is the "study" of "genomic" "sequences" of "strains" and "results" to the creation of a "tree".

A similar observation can be made for the second sample query (i.e., "covid response to weather changes"). For instance, the word "immune", selected from the SKG-based model, is related to the whole query, while word2vec finds a relatedness between the words "defense" and "defence", two spellings of the same noun. Word2vec finds similar words to "change", such as "alter" and "shift", while SKG finds words that describe the *kind* of change like "increase" and "decrease".

query “covid phylogenetic analysis”		**query “covid response to weather changes”**			
word2vec	**SKG**	**word2vec**		**SKG**	
ancestry	genomic	defense	rainfall	immune	climate
phylogenomic	sequence	defence	alter	innate	increase
phylogeographic	genomes	evaluation	shift	induce	decrease
phylogenies	strains	innate	fluctuation	cell	result
ethnographic	tree	meteorological	decline	meteorological	
analisi	study	climate		humid	
	result	ambient		temperature	

Table 3.2: Top expansion terms obtained by the word2vec and the knowledge graph models for two sample queries “covid phylogenetic analysis” and “covid response to weather changes”. The expansion terms for “covid” are not included.

We believe that these differences are due to the way the two models define similarity and relevance.

Experiments on different query expansion strategies with SKGs.

An expanded query can be constructed in many ways depending on the retrieval system’s primary goals. Usually, when synonym terms are added, the query is focused on recall (expanding to include anything relevant) as opposed to precision (ensuring everything included is relevant). Table 3.3 demonstrates results from leveraging minimum match thresholds and percentages, which tilt the scale between precision and recall.

Evaluation Metrics	**BM25-BASE**	**QE-SKG-SNW-BM25**	**QE-SKG-IPRR-BM25**	**QE-SKG-IPNR-BM25**	**QE-SKG-IR-BM25**
Relevant Retrieved	9.955	10.189	9.674	9.824	**10.197**
Recall@1000	41.2	42.9	39.0	39.7	**42.6**
Precision@5	70.8	68.4	**72.8**	71.8	71.2
Precision@10	66.4	66.0	67.5	67.2	**68.2**
Precision@20	62.0	61.8	62.4	61.8	**63.4**
Ndcg@10	60.0	57.6	60.0	59.7	**61.0**
Ndcg@20	57.9	55.2	56.4	56.4	**58.0**
Map	22.0	22.0	22.0	22.0	**23.0**

Table 3.3: Results from different query expansion techniques applied on SKG.

It is interesting that the best results in most metrics are derived from the *query expansion with increased recall* strategy (QE-SKG-IR-BM25). In this case, two synonym terms were required to match the documents, but the original query term is not required to explicitly match. This means that documents which are conceptually similar but don't necessarily contain the original query term are retrieved. Although this method is intended to improve recall it also yields the best results in precision (and ndcg) outperforming all other variations, even the BM25 model. The only case that precision is better is for the *query expansion with increased precision and reduced recall* strategy (QE-SKG-IPRR-BM25) where in order for a document to be considered relevant it must contain at least the 30% of the terms of the query. This has caused an obvious reduction in recall, but it was also "stricter" with the documents that were considered relevant, leading to an increase in precision for the top 5 documents.

Certainly, there exists a boundless array of potential query variations to investigate when refining a query to incorporate richer semantic context. However, the aforementioned strategies should offer a solid understanding of the types of choices at our disposal and the trade-offs we should take into account.

3.5.2 Experiments on Re-Ranking

The results of the different re-ranking models are shown in Table 3.4. We observe that almost all pre-trained S-BERT models are quite effective and outperform standard BM25. The relatively low scores on the S-BERT trained on *ClinicalBERT* can be attributed to the less generic nature of the model. Also, the model parameters have not been fine-tuned.

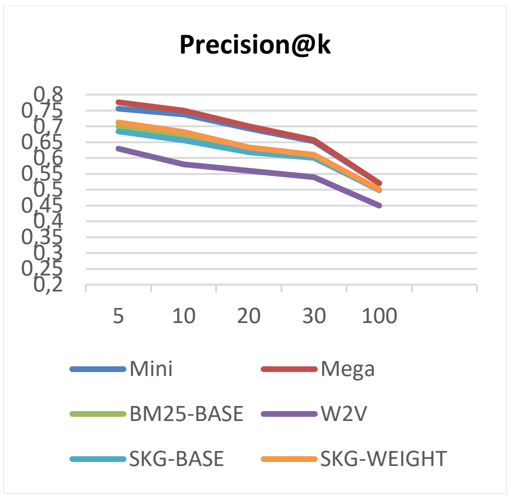
Metrics	**Models**	**BM25-BASE**	**Mega (1, 18)**	**Mega (1, 25)**	**Mini (1, 13)**	**Mini (1, 18)**	**Clinical (1, 18)**
Precision@5		70.8	76.8	**77.6**	75.2	75.6	69.7
Precision@10		66.4	**75.0**	74.2	73.6	73.8	64.0
Precision@20		62.0	69.7	**70.0**	69.2	69.3	61.1
Ndcg@10		60.0	67.4	67.0	67.5	**68.2**	57.0
Ndcg@20		57.9	63.0	**63.6**	63.2	63.4	55.0

Table 3.4: Results of the re-ranking methods. The numbers in the parenthesis are the weights applied to BM25 scores and S-BERT scores.

It should be noted that the models pre-trained on *all-MiniLM-L6-v2* and *all-mpnet-base-v2* have a similar performance. However, the use of *all-mpnet-base-v2* demanded almost triple the time to create the embeddings than *all-MiniLM-L6-v2* model (25 hours as opposed to 11). It's important to strike a balance between the performance of a model and the duration it takes for it to make inferences.

3.5.3 General remarks

Figure 3.1 shows the Precision@k and NDCG@k curves discussed in the previous sections for the models with the best performance. We can observe that the distinction between BM25 and the proposed methods is clearer for small *k*.



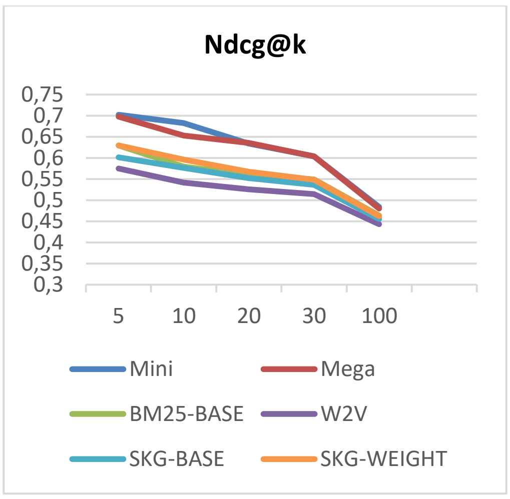


Figure 3.1: Precision@k and NDCG@k curves of the proposed methods against the BM25 baseline for the top 100 documents on TREC-Covid dataset.

In Table 3.7 we present the models with the best performance against ad-hoc retrieval i.e. BM25 model. As already discussed the SKG-based query expansion model outperforms BM25 and word2vec-based query expansion. Re-ranking with transformers outperforms all other models.

System	**R@1000**	**P@20**	**P@10**	**P@5**	**Ndcg@20**
	Ad-hoc retrieval				
BM25	41.21	62.00	66.40	70.00	57.90
	WORD2VEC-based query expansion				
QE-W2V-BM25-CBOW	42.00	59.00	62.00	67.50	54.00
QE-W2VW-BM25-CBOW	42.50	56.00	58.00	63.00	49.00
	SKG-based query expansion				
QE-SKG-SNW-BM25	42.90	61.80	66.00	68.50	55.27
QE-SKG-SW-BM25	**43.20**	61.40	**67.00**	**71.00**	**59.64**
	Re-Ranking				

Mini (1, 18)	41.21	69.30	73.80	75.20	63.46
Mega (1,18)	**41.21**	**70.00**	**75.00**	**77.60**	**63.60**

Table 3.7: Collective presentation of the models with the best performance.

Chapter 4: Related Work

Information retrieval (IR) is the process of finding and retrieving relevant material that satisfies an information need, which could include documents, images, audio files, or other forms of data, from large collections. This material is typically unstructured, encompassing text documents, pixel data in images, and soundwaves in audio files, among others. The fundamental goal of IR is to fulfill a user's information need by identifying and presenting content that is pertinent and valuable.

The evolution of IR has been marked by the introduction of various retrieval models aimed at effectively organizing and ranking information within collections. Among the earliest models are the vector space models, pioneered by Salton et al. in 1975, which represent documents and queries as vectors in a high-dimensional space and calculate similarity scores to rank results. Another significant milestone was the development of probabilistic ranking models by Robertson and Jones in 1976, which introduced probabilistic techniques for ranking documents based on their relevance to a query.

In more recent years, the advent of learning to rank (LTR) models, as proposed by Li in 2014, has further advanced the field of IR by incorporating machine learning algorithms to optimize the ranking of search results based on relevance feedback and user interactions.

The emergence of neural networks has sparked a new wave of innovation in IR, giving rise to the field of neural information retrieval. These neural models leverage deep learning architectures to process and understand complex patterns within textual and multimedia data, leading to improved retrieval performance. Additionally, neural IR has paved the way for novel applications such as question answering systems, multimodal information retrieval, personalized search and recommendation systems, semantic search and conversational search like virtual assistants, chatbots, and voicebased search interfaces, which go beyond traditional keyword-based retrieval to provide direct, context-aware answers to user queries.

Query Expansion Techniques

Embedding-based Approaches

Embedding-based query expansion is a methodology for query expansion that leverages *static embeddings* to identify relevant terms in response to a given query. Static embeddings, such as those generated by *Word2Vec* (Mikolov et al., 2013) and *GloVe* (Pennington et al. (2014), are employed due to their ability to encapsulate semantic similarities between terms. This approach has been explored in various studies.

Diaz et al. (2016) propose a novel approach to embedding-based query expansion, where they employ pre-trained word embeddings to enhance the retrieval effectiveness of information retrieval systems. Their method involves generating an expanded query by including synonyms and related terms of the original query, based on the semantic similarities captured within the embedding space. By making good use of the rich semantic information encoded in the embeddings, their approach aims to address the vocabulary mismatch problem and improve the retrieval performance. Through experimental evaluations on standard IR benchmarks, Diaz et al. demonstrate the effectiveness of their embedding-based query expansion technique in retrieving relevant documents and enhancing the overall retrieval accuracy.

Kuzi et al., (2016) demonstrate a query expansion technique that emphasizes on the Continuous Bag of Words (CBOW) model. The CBOW model, introduced by Mikolov et al. (2013), is a neural network architecture specifically designed to learn distributed representations of words based on their contexts. The CBOW model enables the identification of relevant terms and phrases to expand a given query, thereby improving the retrieval effectiveness of information retrieval systems. Through empirical evaluations on standard IR benchmarks, Kuzi demonstrates the effectiveness of their CBOW-based query expansion approach in retrieving pertinent documents and enhancing overall retrieval accuracy.

Dalton et al. (2019) investigates methods for query expansion customized to multifaceted complex topics, focusing on both local and global approaches. It explores expansion techniques based on words and entities, extending these methods to complex topics by performing fine-grained expansion on different elements of the hierarchical query structure. The results reveal that entity-based expansion methods, especially when combined with local feedback, exhibit significant improvements over word-based models alone. By incorporating both the hierarchical query structure and entity-based expansion, the proposed models achieve over a 20% enhancement in retrieval performance on the TREC Complex Answer Retrieval (CAR) benchmark.

In more recent work, *contextualized* word representation models, such as *ELMo* and *BERT*, are rapidly replacing static embedding models. Naseri et al. (2021) propose the CEQE model. This model builds on the Transformer-based architecture and utilizes contextualized embedding vectors generated by pre-trained models like BERT. Unlike traditional static embedding methods, CEQE includes contextual information from the surrounding text (window size), allowing for more nuanced understanding of word meanings and relationships within the query. By leveraging contextualized representations, CEQE enhances the process of query expansion in information retrieval systems, leading to improved retrieval performance and relevance.

Knowledge Graph-based approaches

In 2012, Google made the term *Knowledge Graph (KG)* widely known through a blog post titled *Introducing the Knowledge Graph: things, not strings*[15], and implemented this approach in their core business, web search. Knowledge graphs provide structured representations of concepts (i.e. entities) and their relationships, enable the incorporation of domain-specific knowledge and can enhance the relevance of retrieved documents. Since then, countless research efforts have been made in the area and KGs are used in a wide variety of applications. Probably, the most visible application of knowledge graphs is semantic search (Balog, 2018). A query contains entities therefore answers to the query can be sourced from the entities of a knowledge graph.

In this vein, KG-based approaches for query expansion involve traversing the graph to find related concepts/entities and adding them to the query (Bollacker et al., 2008; Suchanek et al., 2007).

[15] http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html

KG-based query expansion has been studied for XML data (Zeng et al. 2014), emphasizing strategies to handle missed matches. This involves generating an approximate answer to an original query Q by augmenting its response with content nodes of identical type from a KG, and proposing new keywords to substitute those lacking a match in Q.

Zhang et al. (2017) propose a ranking model to effectively evaluate the similarity of entities to a small number of given seeds (the terms of the query), according to the semantic correlations among entities in knowledge graphs. Extensive experiments on public datasets show that the proposed solution significantly outperforms the state-of-the-art techniques.

Grainger et al. (2016) demonstrates the practical abilities of SKG in the domain of job search. The SKG leverages an inverted index and a complementary uninverted index to represent nodes and edges, providing a layer of indirection between nodes and their corresponding edges. This enables edges to materialize dynamically from underlying corpus statistics, allowing for the discovery of latent relationships between nodes. The performance of the SKG proved impressive, with the ability to swiftly traverse millions of nodes in mere milliseconds on commodity servers, even when factoring in the relatedness score.

Recent advances in knowledge-graph-based research focus on knowledge representation learning (KRL) or knowledge graph embedding (KGE) by mapping entities and relations into low-dimensional vectors while capturing their semantic meanings (Wang et al, 2017), (Lin et al., 2018). Although one can simply measure the relevance of an entity to a query entity by calculating the distance between their vectors, a more promising strategy is to directly learn an embedding model for relevant entity pairs.

Document Re-Ranking with Dense Vectors

Neural ranking models are generally classified into two classes: representation-based models and interaction-based models. Representation-based models focus on independently learning dense vector representations of queries and documents that can be compared to compute relevance via a simple metric such as cosine similarity or inner products. Interaction-based models compare the representations of terms in the query with terms in a document to produce a similarity matrix that captures term interactions. This matrix then undergoes further analysis to arrive at a relevance score. In both cases, models can incorporate many different neural components (e.g., convolutional neural networks and recurrent neural networks) to extract relevance signals. One of the earliest neural ranking models in the deep learning era, the Deep Structure Semantic Model (DSSM) (Huang et al., 2013) constructs character n-grams from an input (i.e., query or document) and passes the results to a series of fully-connected layers to produce a vector representation. At retrieval time, query and document representations can then be compared with cosine similarity.

Interestingly, we are witnessing a resurgence of interest in representation-based approaches, albeit using transformer architectures. Transformers have demonstrated remarkable performance in various natural language processing tasks, including document ranking. Recent studies (Devlin et al., 2018; Yang et al., 2019) have shown the effectiveness of transformer-based models, such as BERT and its variants, in document re-ranking. These models utilize self-attention mechanisms to capture complex relationships between words and sentences, allowing for more accurate document representations.

Biomedical Information Retrieval

In the biomedical domain, document re-ranking plays a crucial role in improving the retrieval of relevant literature. Prior research (Wang et al., 2021; Cohan et al., 2020) has investigated the application of transformer-based models for biomedical information retrieval tasks. These studies have demonstrated the effectiveness of transformer architectures in capturing domain-specific semantics and improving the retrieval accuracy of biomedical documents.

Transformers and BERT have been widely adopted in biomedical research due to their effectiveness in processing natural language data, including medical texts and biomedical literature. (Choi et al., 2019) present ClinicalBERT, a domain-specific variant of BERT fine-tuned on clinical notes. ClinicalBERT demonstrates improved performance on various clinical NLP tasks, including predicting hospital readmissions. In (Lee et al., 2020) BioBERT is introduced as a pre-trained language representation model based on BERT for biomedical text mining tasks. The model is pre-trained on large-scale biomedical corpora and shows improved performance over general-domain BERT in biomedical NLP tasks. In (Yao et al., 2020) a comparison of the performance of BioBERT, ClinicalBERT, and other publicly available embeddings in various biomedical text mining tasks is performed. The authors demonstrate that Biomedical BERT outperforms other embeddings in tasks such as named entity recognition and relation extraction.

These papers provide a glimpse into the diverse applications of transformers and BERT in the biomedical domain, highlighting their effectiveness in tasks such as clinical text mining, disease detection, and biomedical entity recognition.

Chapter 5: Conclusions and Further Work

5.1 CONCLUSIONS

In this study, we investigated various techniques to enhance information retrieval performance, particularly in the context of medical information retrieval using the TREC-COVID dataset. We explored three main avenues: query expansion using word embeddings and semantic knowledge graphs, and re-ranking using transformer-based sentence embeddings. Notably, our findings underscored the significance of semantic knowledge graphs in providing contextually rich expansions for queries, while transformer-based re-ranking demonstrated remarkable improvements over traditional BM25 methods. These discoveries pave the way for future advancements in medical information retrieval, offering promising avenues for further research and development.

5.2 FURTHER WORK

Looking ahead, a promising direction for future research lies in the fine-tuning of transformer models. By fine-tuning transformer architectures on large-scale medical corpora, we can harness the power of contextual embeddings to capture intricate semantic relationships and domain-specific terminology more effectively. This fine-grained tuning process holds the potential to yield highly specialized models capable of delivering tailored search results with enhanced accuracy and relevance, thereby advancing the frontier of medical information retrieval to new heights.

Bibliography

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. CoRR, vol. abs/1706.03762, 2017.
2. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. An Introduction to Information Retrieval, Cambridge University Press, 2009, p. ?.
3. Dalton, J., Naseri, S., Dietz, L., & Allan, J. (2019). Local and global query expansion for hierarchical complex topics. In European conference on information retrieval (pp. 290–303).
4. Edward, Mohammad Taha Bahadori, Andy Schuetz, Walter F. Stewart, Jimeng Sun. ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission. Journal of the American Medical Informatics Association (JAMIA), 2019.
5. Ellen M Voorhees and Kirk Roberts. On the Quality of the TREC-COVID IR Test Collections. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2021.
6. Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection. SIGIR Forum, June, 2020.
7. F. M. Y. Yao, L. L. Coelho, S. D. B. Barbosa. Biomedical BERT outperforms other publicly available embeddings in biomedical text mining tasks. arXiv:2004.14535, 2020.
8. Fernando Diaz, Bhaskar Mitra, and Nick Craswell. 2016. Query expansion with locally-trained word embeddings. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 367–377. https://doi.org/10.18653/v1/P16-1035
9. Grainger, T., Aljadda, K. & Korayem, M. (2016). The Semantic Knowledge Graph: A Compact, Auto-Generated Model for Real-Time Traversal and

Ranking of any Relationship within a Domain. In IEEE International Conference on Data Science and Advanced Analytics (DSAA)

10. Hiteshwar Kumar Azad, Akshay Deepak. Query expansion techniques for information retrieval: A survey. Information Processing & Management, 2019.
11. Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, Volume 36, Issue 4, February 2020.
12. Jimmy Chen and William R Hersh. Analysis of System Features Used in the TREC-COVID Information Retrieval Challenge. Journal of Biomedical Informatics, 117:103745, 2021.
13. Joan Guisado-Gámez, David Dominguez-Sal, Josep-LLuis Larriba-Pey. Massive Query Expansion by Exploiting Graph Knowledge Bases. arXiv:1310.5698 [cs.IR]. 2013.
14. Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, Volume 36, Issue 4, February 2020.
15. Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R Hersh. TREC-COVID: Rationale and Structure of an Information Retrieval Shared Task for COVID-19. Journal of the American Medical Informatics Association, 27(9):1431–1436, 2020.
16. Kuzi, S., Shtok, A., & Kurland, O. (2016). Query expansion using word embeddings. In Proceedings of the 25th acm international on conference on information and knowledge management.
17. Lin, Y., Han, X., Xie, R., Liu, Z., and Sun, M., Knowledge representation learning: A quantitative review. arXiv preprint arXiv:1812.10901, 2018.
18. Naseri, S., Dalton, J., Yates, A., & Allan, J. (2021, March). CEQE: Contextualized embeddings for query expansion. In European Conference on Information Retrieval (pp. 467–482). Springer, Cham.

19. Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP 2019.

20. Onal, K. D., Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H.-L. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. de Rijke, and M. Lease. (2017). "Neural information retrieval: at the end of the early years". Information Retrieval Journal. 21(2-3): 111–182. doi: 10.1007/s10791-017-9321-y.

21. P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In Proceedings of 22nd International Conference on Information and Knowledge Management (CIKM 2013), pages 2333–2338, 2013.

22. Papagiannopoulou, E. and G. Tsoumakas. (2018). "Local word vectors guiding keyphrase extraction". Information Processing and Management. 54(6): 888–902. doi: 10.1016/j.ipm.2018.06.004.

23. Park, S. and C. Caragea. (2020). "Scientific Keyphrase Identification and Classification by Pre-Trained Language Models Intermediate Task Transfer Learning". In: Proceedings of the 28th International Conference on Computational Linguistics. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main .472.

24. Pennington, J., R. Socher, and C. Manning. (2014). "Glove: Global Vectors for Word Representation". In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics. doi: 10.3115/v1/d14-1162.

25. Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications," IEEE TKDE, vol. 29, no. 12, pp. 2724–2743, 2017

26. S. Zeng, Z. Bao, T.W. Ling, H. Jagadish, G. Li. Breaking out of the mismatch trap. In: ICDE, pp 940–951, (2014).

27. Stephen Robertson & Hugo Zaragoza (2009). The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information

Retrieval. 3 (4): 333–389. CiteSeerX 10.1.1.156.5282. doi:10.1561/1500000019. S2CID 207178704.

28. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean (2013). Distributed Representations of Words and Phrases and their Compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems. Vol. 2. Lake Tahoe, NV, pp. 3111–3119.
29. T. Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.
30. Thomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Advances in Neural Information Processing Systems 26, pages 3111–3119. 2013.
31. Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. arXiv preprint arXiv:1309.4168, 2013.
32. Trey Grainger et al. The Semantic Knowledge Graph: A Compact, Auto-Generated Model for Real-Time Traversal and Ranking of any Relationship within a Domain. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (2016): 420-429.
33. Wang, Lucy Lu and Lo, Kyle and Chandrasekhar, Yoganand and Reas, Russell and Yang, Jiangjiang and Burdick, Doug and Eide, Darrin and Funk, Kathryn and Katsis, Yannis and Kinney, Rodney Michael and Li, Yunyao and Liu, Ziyang and Merrill, William and Mooney, Paul and Murdick, Dewey A. and Rishi, Devvret and Sheehan, Jerry and Shen, Zhihong and Stilson, Brandon and Wade, Alex D. and Wang, Kuansan and Wang, Nancy Xin Ru and Wilhelm, Christopher and Xie, Boya and Raymond, Douglas M. and Weld, Daniel S. and Etzioni, Oren and Kohlmeier, Sebastian. CORD-19: The COVID-19 Open Research Dataset. Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020. July 2020.
34. Zhang, X., Chen, Y., Chen, J., Du, X., Wang, K., & Wen, J. (2017). Entity Set Expansion via Knowledge Graphs. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval

Appendices

```
aggregation_query = {
                "query": {
                    "match": {
                        "text_field": word x
                    }
                },
                "aggs": {
                    "significant_terms": {
                        "significant_text": {
                            "field": "text_field",
                            "background_filter": {
                                "match_all": {}
                            }
                        }
                    }
                }
            }
            aggregation_result = es.search(index=index_name, 
body=aggregation_query)
…
            significant_terms = 
aggregation_result["aggregations"]["significant_terms"]["buckets"]
            for term in significant_terms:
                if term['score']>17:
                    expanded_keywords.append(term['key'])
                    print(word + "   adding : "  +term['key'])
```

Code Snippet 1 Significant Terms aggregation

The code Snippet 1 shows how we expand the query using the significant terms aggregation which materializes the SKG method. We only keep those terms that have higher score than 17. This code implements the QE-SKG-SNW-BM25 approach.

```
for term in significant_terms:
                if term['score'] > 3:
                    if term in covid_list:
                        boost=term['score']/200.0
                    boost = term['score'] / 80.0  # Adjust as needed
                    expanded_keywords.append(f"{term['key']}^{boost}")
```

Code Snippet 2 SKG-Weighted queries

The code Snippet 2 shows the differences between the QE-SKG-SNW-BM25 and QE-SKG-SW-BM25 approach. In the code above we expand the query with more terms but multiply them with the similarity score they have. Also we check if a term is in the “covid_list” and if so we adjust the weight as shown.

```
def find_similar_words(word, word2vec_model, top_max_n=2:
    try:
        word_vector=word2vec_model.wv[word]
        similar_words =
word2vec_model.wv.similar_by_vector(word_vector, topn=top_max_n)
        similar_words = [(token, score) for token, score in
similar_words if score >= min_sim]
        similar_words = [token for token, _ in similar_words if token
!= word]
        return similar_words
```

Code Snippet 3 W2V expansion

The code Snippet 3 retrieves the similar words to the input query word based on their similarity score. It keep only similar words with a similarity score above a certain threshold (min_sim). Finally, it returns a list of similar words excluding the input word itself.

```
def combine_scores(bm25_score, query_embedding, document_embedding,
bm25_weight=1.0, bert_weight=x(10-25):
    bert_score = float(STutil.pytorch_cos_sim(query_embedding,
document_embedding).numpy())
    combined_score = bm25_weight * bm25_score + bert_weight *
bert_score
    return combined_score
```

Code Snippet 4 Combination of Bert anM@5.

The code Snippet 4 shows the function that combines the relevance score of the 2 approaches after first calculating the BERT score using the cosine similarity between the query and document embedding. Then it combines the BM25 and Bert scores using the provided weights and returns the combined score.