EVALUATING THE USE OF NEURAL RANKING METHODS IN SEARCH ENGINES

SİNİRSEL SIRALAMA YÖNTEMLERİNİN ARAMA MOTORLARINDA KULLANIMININ DEĞERLENDİRİLMESİ

Ömer ŞAHİN

Prof. Dr. İlyas ÇİÇEKLİ
Supervisor

Dr. Gönenç ERCAN
Co-Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
In Computer Engineering

2022

ABSTRACT

EVALUATING THE USE OF NEURAL RANKING METHODS IN SEARCH ENGINES

Ömer ŞAHİN

Master of Sciences, Department of Computer Engineering

Supervisor: Prof. Dr. İlyas ÇİÇEKLİ

Co-Supervisor: Dr. Gönenç ERCAN

January 2022, 77 pages

A search engine strikes a balance between effectiveness and efficiency to retrieve the best documents in a scalable way. Recent deep learning-based ranker methods prove effective and improve state of the art in relevancy metrics. However, unlike index-based retrieval methods, neural rankers like BERT do not scale to large datasets. In this thesis, we propose a query term weighting method that can be used with a standard inverted index without modifying it. Using a pairwise ranking loss, query term weights are learned using relevant and irrelevant document pairs for each query. The learned weights prove to be more effective than term recall values previously used for the task. We further show that these weights can be predicted with a BERT regression model and improve the performance of both a BM25 based index and an index already optimized with a term weighting function. In addition, we examine document term weighting methods in the literature that work by manipulating term frequencies or expanding

documents for document retrieval tasks. Predicting weights with the help of contextual knowledge about document instead of term frequencies for documents terms significantly increase retrieval and ranking performance.

Keywords: Information Retrieval, Passage Ranking, Term Weighting, Pairwise Ranking Optimization

ÖZET

SİNİRSEL SIRALAMA YÖNTEMLERİNİN ARAMA MOTORLARINDA KULLANIMININ DEĞERLENDİRİLMESİ

Ömer ŞAHİN

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Danışman: Prof. Dr. İlyas ÇİÇEKLİ

Eş Danışman: Dr. Gönenç ERCAN

Ocak 2022, 77 sayfa

Bir arama motoru, en alakalı belgeleri ölçeklenebilir bir zamanda alabilmeli, etkinlik ve verimlilik arasında bir denge kurmalıdır. Son zamanlardaki derin öğrenme tabanlı sıralayıcı yöntemlerinin etkili olduğu kanıtlanmıştır ve alaka ölçütlerinde en son teknolojiyi oluşturmaktadır. Ancak, dizin tabanlı alma yöntemlerinin aksine, BERT gibi sinirsel sıralayıcılar büyük veri kümelerine ölçeklenemez. Bu tezde, standart bir ters indekslemeyi değiştirilmeden kullanılabilecek bir sorgu terimi ağırlıklandırma yöntemi öneriyoruz. Sorgu terim ağırlıkları, ikili sıralama kaybı kullanılarak her sorgu için alakalı ve alakasız belge çiftleri kullanılarak eğitilir. Öğrenilen ağırlıkların, bu görev için daha önce kullanılan terim hatırlama değerlerinden daha etkili olduğu kanıtlanmıştır. Ayrıca, bu ağırlıkların bir BERT regresyon modeli ile tahmin edilebileceğini ve hem BM25 tabanlı bir indeksin hem de bir terim ağırlıklandırma fonksiyonu ile halihazırda optimize edilmiş bir indeksin performansını iyileştirdiğini gösteriyoruz. Ek olarak, belge alma görevleri için terim sıklıklarını değiştirerek veya belgeleri genişleterek çalışan literatürdeki

belge terimi ağırlıklandırma yöntemlerini inceliyoruz. Belge terimleri için terim frekansları yerine belge hakkındaki bağlamsal bilginin yardımıyla ağırlıkları tahmin etmek, alma ve sıralama performansını önemli ölçüde artırır.

Anahtar Kelimeler: Bilgi Getirme, Pasaj Sıralama, Terim Ağırlıklandırma, İkili Sıralama Optimizasyonu

ACKNOWLEDGEMENTS

I am grateful to my supervisor Prof. Dr. İlyas Çiçekli and my co-supervisor Dr. Gönenç Ercan for guiding me in this journey and illuminating the way I walkthrough.

I thank valuable lecturers of Hacettepe University, Department of Computer Engineering, for the courses that improve my skills and experiences in the field when I studied for my master's.

In addition, I would like to thank my family for supporting me unsparingly while I am working on accomplishing my degree of master.

CONTENTS

ABSTRACT **i**

ÖZET **iii**

ACKNOWLEDGEMENTS **v**

CONTENTS **vii**

LIST OF FIGURES **xi**

LIST OF TABLES **xiii**

LIST OF SYMBOLS **xv**

LIST OF ABBREVIATIONS **xvii**

1 INTRODUCTION **1**
1.1 Motivation and Scope 1
1.2 Contribution 2
1.3 Thesis Organization 3

2 LITERATURE REVIEW **5**
2.1 Ranking Function 5
2.2 Relevance Feedback 7
2.3 Term Weighting 7
2.4 Learning to Rank 9

3 LANGUAGE MODELS **11**
3.1 Transformer 11
3.1.1 Encoder and Decoder 11
3.1.2 Attention 13
Scaled Dot-Product Attention 13
Multi-Head Attention 14
3.1.3 Position-wise Feed-Forward Networks 14
3.1.4 Positional Encoding 15

3.2 Bidirectional Encoder Representations from Transformers 15
3.2.1 Model Architecture 16
3.2.2 Input/Output Representations 16
3.2.3 Pre-Training 16
Masked Language Model 16
Next Sentence Prediction 17
3.2.4 Fine-tuning 17
3.3 Text to Text Transfer Transformer 17
3.3.1 Model Architecture 18
3.3.2 Input/Output Representations 18
3.3.3 Pre-Training 19
3.3.4 Fine-Tuning 19

4 SEARCH ENGINE 21
4.1 Lucene 22
4.1.1 Text Index 22
4.1.2 Query Search 23

5 RELEVANCE FEEDBACK 25
5.1 Term Recall Based Term Weighting 26
5.2 Pairwise Ranking Loss Based Term Weighting 27
5.2.1 Min-Max Normalization 29
5.2.2 Minimization Absolute Values of Negative Weights 30
5.2.3 Non-Negative Weight Constraint 30
5.2.4 Pair Selection 31

6 TERM WEIGHTING 33
6.1 Query Term Weighting 33
6.1.1 DeepTR 33
6.1.2 DeepCT-Query 35
6.1.3 Term Weight Prediction Model 36
6.2 Document Term Weighting 38
6.2.1 DeepCT - Index 39
6.2.2 TextRank 40
6.2.3 Doc2Query 41
6.2.4 DocTTTTTQuery 41

7 EVALUATION **43**
7.1 Recall 43
7.2 Mean Average Precision 44
7.3 Mean Reciprocal Rank 44
7.4 Normalized Discounted Cumulative Gain 44
7.5 Paired t-Test 45

8 EXPERIMENTS **47**
8.1 Dataset 47
8.2 Configurations 49
8.3 Methods to Evaluate 50

9 RESULTS **53**
9.1 Number of Pair for Pairwise Term Weight Optimization 54
9.2 Query Term Weighting 55
9.2.1 Term Recall and Pairwise Term Weight Optimization 55
9.2.2 Term Weight Estimation 57
9.2.3 Combining with Index Term Weighting 59
9.3 Document Term Weighting 61
9.3.1 Term Frequency Models 61
9.3.2 Term Expansion Models 62
9.4 Phrase Search 63
9.4.1 Maximum Length of Phrases 63
9.4.2 Phrase Weighting 64
9.5 Overall Results 66

10 CONCLUSION **69**

REFERENCES **71**

LIST OF FIGURES

3.1 Transformer Network Architecture 12
3.2 Attention Mechanism of Transformer 13

6.1 BERT Query Term Weighting Model Architecture 38
6.2 Doc2Query Document Expansion Model 41

8.1 MS MARCO Passage-Term Histogram 48
8.2 MS MARCO Query-Term Histogram 49
8.3 MS MARCO Query-Relevant Passage Histogram 49

LIST OF TABLES

5.1 Relevant and irrelevant pair for a query. 32

8.1 The number of passages and queries in MS MARCO collection. . . 47
8.2 BM25 parameter configurations for methods. 50

9.1 Number of documents to optimize pairwise model. 54
9.2 Query term weighting with oracle methods using the document relevance information. MRR, MAP, and RECALL metrics evaluation. 56
9.3 Query term weighting with oracle methods using the document relevance information. NDCG metric evaluation. 56
9.4 Query term weighting using estimated term weights. MRR, MAP, and RECALL metrics evaluation. 58
9.5 Query term weighting using estimated term weights. NDCG metric evaluation. 59
9.6 Combined effect of query term weighting with index term weighting (DeepCT-Index). MRR, MAP, and RECALL metrics evaluation. 60
9.7 Combined effect of query term weighting with index term weighting (DeepCT-Index). NDCG metric evaluation. 60
9.8 Document term weighting. MRR, MAP, and RECALL metrics evaluation. 61
9.9 Document term weighting. NDCG metric evaluation. 62
9.10 Document term expansion. MRR, MAP, and RECALL metrics evaluation. 63
9.11 Document term expansion. NDCG metric evaluation. 63
9.12 Maximum length of the phrases. 64
9.13 Term Weighting for Phrase Query. MRR, MAP, and RECALL metrics evaluation. 65
9.14 Term Weighting for Phrase Query. NDCG metric evaluation. 65
9.15 Overall results of query and document term weighting. 67

LIST OF SYMBOLS

Q	Query
D	Document
D_{Rel}	Relevant document
D_{Irrel}	Irrelevant document
D_K	Top K documents
q_i	ith term of the query
$avgdl$	Average document length
D_q	Set of relevant documents
$D_{q,t}$	Set of relevant documents that contain term t
Q_d	Set of queries that address the document
$Q_{d,t}$	Set of queries that address the document and contain term t
$\vec{w}$	Weight vector of query terms
w_i	Weight of ith query term
W_{max}	Maximum term weight in $\vec{w}$
W_{min}	Minimum term weight in $\vec{w}$
$\vec{x}$	Feature vector of the relevant document
$\vec{y}$	Feature vector of the irrelevant document
α	Minimum distance between relevant and irrelevant documents, margin

LIST OF ABBREVIATIONS

NLP	**N**atural **L**anguage **P**rocessing
TF	**T**erm **F**requency
IDF	**I**nverse **D**ocument **F**requency
BoW	**B**ag **o**f **W**ords
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
T5	**T**ext **t**o **T**ext **T**ransfer **T**ransformer
MAP	**M**ean **A**verage **P**recision
MRR	**M**ean **R**eciprocal **R**ank
NDCG	**N**ormalized **D**iscounted **C**umulative **G**ain
qid	**Q**uery **ID**
pid	**P**assage **ID**
Min-Max	**Min**inimum-**Max**imum Normalization
Min-Abs-Neg	**Min**imization **Abs**olute Values of **Neg**ative Weights
Non-Neg	**Non-Neg**ative Weight Constraint

1. INTRODUCTION

1.1 Motivation and Scope

A typical search engine retrieves the "best" matching documents to a user query. Its effectiveness is measured through how accurately it ranks documents against the query. On the other side, the retrieval algorithm must scale to billions of documents and thus must be efficient. Recent research show that contextualized word embeddings like BERT [6] are effective in retrieval and outperform the long-standing baseline BM25 [34] significantly. Unfortunately, processing all documents with a ranker like BERT is not efficient and scalable. Recent studies try to address this problem by striking a balance between effectiveness and efficiency using multi-stage retrieval systems [25, 3].

In multi-stage search engines, the first stage is to create a subset of documents by selecting the documents that are possibly related to the query. The second stage is re-ranking the candidate documents according to their relevance with the query. In the second stage, more complex learning-to-rank algorithms re-rank the candidate documents chosen in the first stage.

As an alternative solution, it is possible to process only the query with BERT and transfer information via term weights to an efficient index-based retrieval method like BM25. This allows the retrieval system to scale to large datasets but still be able to capture the importance of the terms using rich semantic information captured in BERT embeddings. Furthermore, a readily available standard inverted index-based retrieval system like Lucene[1] can be used without any modifications.

Frequency-based ranking algorithms such as BM25 use corpus statistics to determine the importance weights for terms. A term appearing in a large portion of the documents is deemed to be less important, while a normally less frequent term is assigned a large weight. Inverse document frequency (IDF) is an effective weighting scheme that incorporates this to BM25. BM25 can be further improved when some form of relevance feedback is available, i.e., when relevant documents to the query are known. The proportion of relevant documents a term appears in is used as the weighting factor. We will refer to this as the Term Recall value of the

[1] Apache Lucene search engine - lucene.apache.org

term. While this improves the retrieval effectiveness, the term recall values are specific to the query and do not generalize to a diverse set of queries.

Term recall based weights are effective in optimizing recall at high cutoff values, as terms appearing in all relevant documents are boosted. A term appearing in all relevant documents will be assigned a high weight value, even though it is also commonly used in irrelevant documents. A stop word appearing in all documents will be assigned a high weight. In this thesis, we investigate the use of pairwise ranking loss to learn the term weights of each query as a replacement for term recall values. This can potentially adjust term weights to better distinguish relevant documents from irrelevant ones. Furthermore, we continue to show that these learned weights can be predicted by the BERT model and yield performance improvements over term recall based weights.

1.2 Contribution

To increase effectiveness and maintain efficiency simultaneously, most of the approaches in the literature do index-time term weighting, which manipulates documents before the index. Considering the size and accessibility of the indices, re-indexing might not be feasible or possible. In other respects, term weights can also be applied in search-time by preprocessing the query with a deep learning model. In such a case, term weighting for query terms is an effective and efficient way to fetch relevant documents for the query. For this purpose, we focus on query term weighting that aims to estimate optimum weights for query terms by optimizing a pairwise ranking loss to achieve higher scores for relevant documents than irrelevant ones and propose a BERT regression model to predict desired target term weights.

We propose a new relevance feedback method that learns optimum term weights for a query to retrieve relevant documents in the top ranks. The pairwise loss-based optimization method tries to find the best coefficients for query terms that boost term contribution in BM25. When boosting terms according to importance fetching relevant documents, it has to not tend to favor irrelevant documents. The proposed relevance feedback method called pairwise term weight optimization generalizes over hard instances of relevant and irrelevant pairs to choose relevant documents instead of irrelevant ones. The optimization highlights terms that

express the query and relevant document and reduce the effect of the generic terms that are insignificant for relevant documents.

Pairwise term weight optimization runs supervised, which means it requires labeled data for the query. To boost query term weights in the search time, we propose BERT based regression model that takes term and the query as input, and the model estimates target term weight which can be term recall or pairwise optimized ones. The proposed BERT regression model allows to phrase weighting like term weighting as distinct from the term estimation frameworks in the literature thanks to the term-query input design. The BERT regression model is trained with the proposed relevance feedback as offline, and with the minor addition of inference time for query term weight estimation in search time, better retrieval and ranking performances are obtained.

Besides our proposed relevance feedback and term weight estimation methods, we bring together a comparison of the query and document term weighting models. We evaluate different approaches for query and document term weighting schemes and investigate the neural network models they use for representing natural language. In addition to the approaches for the studied task in the literature, we present research for relevance feedback and language models.

1.3 Thesis Organization

In the next chapter, we review the literature for seeking the history of ranking functions, relevance feedback, term weighting, and recent learning-to-rank algorithms. The development of the probabilistic ranking functions is investigated, and BM25, which is the most known probabilistic weighting scheme, is used in experiments. Probabilistic relevance feedback method and the different term weighting frameworks are evaluated, and some of term weighting frameworks use the examined relevance feedback method. Learning-to-rank approaches are defined, and recent researches about learning-to-rank algorithms are inspected.

In the chapter of Language Models, Transformer based language models are given. The network architectures of these language models are represented. Pretraining and fine-tuning procedures are explained. The given language models are used by evaluated term weighting models and the proposed term estimation model.

In Search Engine chapter, logic of a typical text search engine is explained. The capabilities of an open-source text search engine called Lucene are given. Text indexing and query searching processes are detailed.

In the chapter of Relevance Feedback, the usage of relevance feedback methods in the ranking function is explained. The existing relevance feedback method called Term Recall is presented, and a pairwise-loss-based optimization method for optimum relevance feedback is proposed.

In Term Weighting chapter, query and document term weighting frameworks are given. Query term boosting methods and the proposed BERT-based term weight estimation model is specified. Document term weighting models that by manipulating term frequencies or expanding documents are indicated.

In the following chapters, evaluation metrics for retrieval and ranking performance are given. The dataset for retrieval task and configurations of the test setup is remarked. In the result chapter, the results of all experiments are explained for different metrics and different test cases.

At the end of the thesis, final thoughts and evaluations are given as a conclusion of the thesis.

2. LITERATURE REVIEW

The text ranking is an old topic. Indexing documents and searching something in that indexed documents done for years. In this chapter, text retrieval and ranking techniques in the literature are reviewed in several titles. In the first part, the ranking schemes and probabilistic weighting schemes are investigated. In the next part, relevance feedback is looked over for retrieval performance. In the next part, query and document term weighting methods are explored. Finally, we inform about learning-to-rank algorithms.

There is a well-known statistical relevance feedback method called term recall and various term weighting schemes that use term recall in literature. Most term weighting models focus on document term weighting, and there are fewer query term weighting models. Furthermore, there is no study to use query and document term weighting together. There is a work area for new relevance feedback and evaluating the combination of query and document term weighting approaches in the literature.

2.1 Ranking Function

One of the most known probabilistic weighting schemes is BM25 [34]. In BM25, BM stands for "Best Match". The probabilistic weighting schemes rely on the exact term matching. To calculate the relevance score for a document with respect to the input query, there are must be common terms between both of them. Consequently, the cumulative relevance scores of each term give the document relevance score.

The traditional probabilistic term weighting scheme can be formulated as follow [31, 32]:

$$\frac{(k_3+1)q}{k_3+q} \times \frac{(k_1+1)f}{k_1L+f} \times \frac{log(r+0.5)(N-n-R+r+0.5)}{(n-r+0.5)(R-r+0.5)} \tag{2.1}$$

where k_1 and k_3 are constant, which helps to scale the weights, q and f are the term frequencies, q is in the query and f is in the document. In the equation, N is the number of total documents in the collection, the number of documents that contain the term is shown as n. The number of relevant documents is R and the

number of relevant documents that contain the term is r. The normalized document length is computed as the length of this document divided by the average document length of all documents in the collection, and the normalized document length is represented as L.

BM11 uses the probabilistic term weighting equation (Equation 2.1) with additional value to sum with cumulative document score. The following equation shows the additional query information:

$$k_2 n_q \frac{1-L}{1+L} \tag{2.2}$$

where k_2 is another constant and n_q is the number of term in query, in other words, length of the query.

BM15 is the same as BM11 except $k_1 L + f$ changed with the $k_1 + f$ in probabilistic weighting scheme (Equation 2.1).

BM25 is the combination of BM11 and BM15 with a scaling factor. Overall relevance score for a document is calculated as follow [43]. The detailed equation of BM25 at Chapter 5, Relevance Feedback.

$$BM25(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot TF(D,q_i) \tag{2.3}$$

BM25F [40, 41] is a variation of BM25 that handles the contribution of a field when the document consists of several fields such as headlines, contents.

BM25+ [33] is an extension for BM25 to dissolve an incompleteness in BM25. As a consequence of the incompleteness, the long documents that contain the term may be scored the same as the shorter documents that do not have the term due to normalization of the term frequencies by the length of the document. To solve this problem, an additional free parameter δ added to $TF(D,q_i)$ as distinct from BM25. BM25 equation is changed to the following equation:

$$BM25(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot [TF(D,q_i) + \delta] \tag{2.4}$$

where δ is an additional free parameter that equals 1 in default.

2.2 Relevance Feedback

Terms have different contributions to the relevance score. The term frequencies do not involve any contextual information to retrieve a document. A relevance feedback metric can be defined when knowing the relevance between queries and documents in the collection.

A probabilistic relevance feedback is defined by Mogotsi [20]. Assume that, for each term x_t in the query, there is constant p_t that is probability estimation. Set of user judgment relevant documents is defined as $R = \{d : R_{d,q} = 1\}$. When relevant and irrelevant document sets are big enough, the relevance feedback p_t for term t is shown as follow:

$$p_t = \frac{|VR_t|}{|VR|} \tag{2.5}$$

where VR_t is the set of relevant documents that contain term x_t, and VR is the set of all relevant documents. With smoothing factor, the relevance feedback for term x_t equals to following equation:

$$p_t = \frac{|VR_t| + \frac{1}{2}}{|VR| + 1} \tag{2.6}$$

This probabilistic relevance feedback can be applied to the document-side. The relevance probability is estimated by replacing with each other the query and document in the assumption. The document-side relevance probability is calculated as the same but this time VR_t is the set of queries that address the document and contain term x_t and VT is the set of queries that address the document, and x_t is the tth term of the document, instead of a query.

The probabilistic relevance feedback without smoothing factor is called Term Recall and it is used in retrieval frameworks [49, 5] to obtain more knowledge than the statistical TF-IDF values.

2.3 Term Weighting

Term frequencies and inverse document frequencies are statistical relevance feedback about a term that is independent of the input query in a basic way. Term

weighting is more accurate relevance feedback about terms for retrieval and ranking relevant documents by replacing term frequency with estimated term weight.

To realize this purpose, there are several methods such as estimating target relevance feedback by contextual embeddings, defining term importance by term co-occurrence, or predicting possible queries that address to the document and expand the document with these possible queries. Some of these methods can be applied in both document and query terms.

DeepTR is a query term weighting method that boosts the score of the term in the ranking function proposed by Zheng and Callan [49]. DeepTR tries to estimate term recall [20] relevance feedback by using the distance between the term and the query as a feature vector. Terms are represented in the feature space by Word2Vec [19] word embeddings. The query is the average of the word vectors that compose the query itself.

Dai and Callan [5] proposed a contextual term weighting framework called DeepCT for document and query terms. The contextual knowledge about text extracted with BERT [6] language model and a regression layer tries to predict target relevance feedback which is term recall [20]. The pre-trained BERT language model is fine-tuned for the task that estimates term recall values with an additional regression layer.

TextRank is a graph-based keyword extraction model proposed by Mihalcea and Tarau [18]. TextRank uses the PageRank [1] algorithm to find the most important terms in the document by using their co-occurrence matrix. Relevance feedback of the terms is defined by the PageRank algorithm in Text Rank. Most co-occurrence terms in the documents are the most important terms to represent the documents and the terms that co-occurrence with most important terms is important as well in regard to the PageRank algorithm.

Document expansion models change term frequencies by adding terms and reduce term mismatching between query and document by adding new terms to the document. Doc2Query [24] and its follow-up work DocTTTTTQuery [23] models expand documents by estimating queries that address to the document. Both models use sequence-to-sequence [42] neural network architecture that takes documents as input sequence and predicts queries as output sequence. Doc2Query model is trained end-to-end to estimate possible queries for

documents. DocTTTTTQuery model is fine-tuned for query estimation task from Text-to-Text Transfer Transformer model trained as a generative language model for different tasks with rich text data by Raffel et al. [30].

A more recent research, DeepImpact [17] combines document expansion by DocTTTTTQuery [23] and term weighting together. The enriched documents with the help of document expansion by DocTTTTTQuery reduce word mismatching and the contextual document term weighting helps to get better ranking and retrieving performance.

2.4 Learning to Rank

Learning-to-Rank algorithms use three main approaches that are point-wise, pair-wise, and list-wise to rank documents, and focus on representation or interaction of query and document.

The learning-to-rank approaches can be listed as follows:

- Point-wise approach learns to rank documents on a single document and input query. The point-wise model calculates the relevance score for each document.
- The pair-wise approach learns which of the document pairs is more relevant to the input query. The model compares the documents in pairs according to their relevance to the query and returns the more relevant document. RankSVM [14] and RankBoost [7] use the pair-wise approach to rank documents by judging which documents in the pair are more relevant with the input query.
- The list-wise approach learns to rank the set of documents according to their relevance to the input query. The model returns an ordered list of documents. ListNet [4], AdaRank [47] and LambdaMart [2] apply the list-wise approach. The set of documents is ranked as a whole in a single run.

Representation-focused learning-to-rank algorithms extract a representative feature vector for the query and the document individually and relevance scores are measured by using the similarity between representation vectors of query and document pair. DSSM [12], CDSSM [39], ARC-I [11], and SQA [36] learning-to-rank algorithms focus on the representation of document and query.

Interaction-focused learning-to-rank algorithms try to match the query and document feature patterns according to the interaction of the query and the document. The more matching patterns the higher the relevance between the query and the document. DRMM [9], ARC-II [11], MatchPyramid [26], Match-SRNN [45], DeepRank [27] learning-to-rank algorithms focus on the interaction between document and query.

Zamani et al. [48] proposed a sparse representation method for documents in the collection. The sparse representation vectors are used to index documents by inverted index [50] structure. The queries are represented as sparse vectors in the same space. The matched non-zero values between query and documents provide document retrieval and the similarity of sparse vectors gives the relevance score.

Recent research BERT-based ranking models like Passage Re-ranking [22], TF-Ranking [10], ColBERT [15] and COIL [8] show that contextualized word embeddings like BERT [6] are effective in retrieval and outperform the long-standing baseline BM25 [34] significantly.

3. LANGUAGE MODELS

3.1 Transformer

Transformer [44] is a model that uses the attention mechanism to detect global dependencies between input and output sequences instead of recurrence. On the machine translation task, Transformer achieved better quality success and the transformer model needs less time for training significantly due to more parallelization than recurrent models.

In many sequence transduction tasks, the main objective is finding relevance between tokens in the long sequence range. Self-Attention provides to learn these dependencies in a long-range by creating combinations of the tokens in the sequence. Additionally, self-attention layers work faster than recurrent layers, if the sequence length is less than the representation dimensionality.

Transformer consists of encoder and decoder architecture, both encoder and decoder use stacked self-attention and point-wise fully connected dense layers. The model architecture of Transformer is given in Figure 3.1.

3.1.1 Encoder and Decoder

The encoder part of Transformer is stacked N identical layers that consist of two sub-layers. The first one of the sub-layers is a multi-head self-attention mechanism and the second one is a position-wise feed-forward network that fully connected. There are residual connections around sub-layers which are followed by a normalization layer.

The decoder of Transformer is stacked N identical layers similar to the decoder. The decoder has one more sub-layer between multi-head attention and position-wise feed-forward block. The additional sub-layer applies multi-head attention to the output of the encoder stack. As in the encoder, there are residual connections around sub-layers and after that, a normalization layer. In the decoder stack, the self-attention sub-layer is adjusted to ensure prediction at position i rely on the known output that position is less than i by masking.

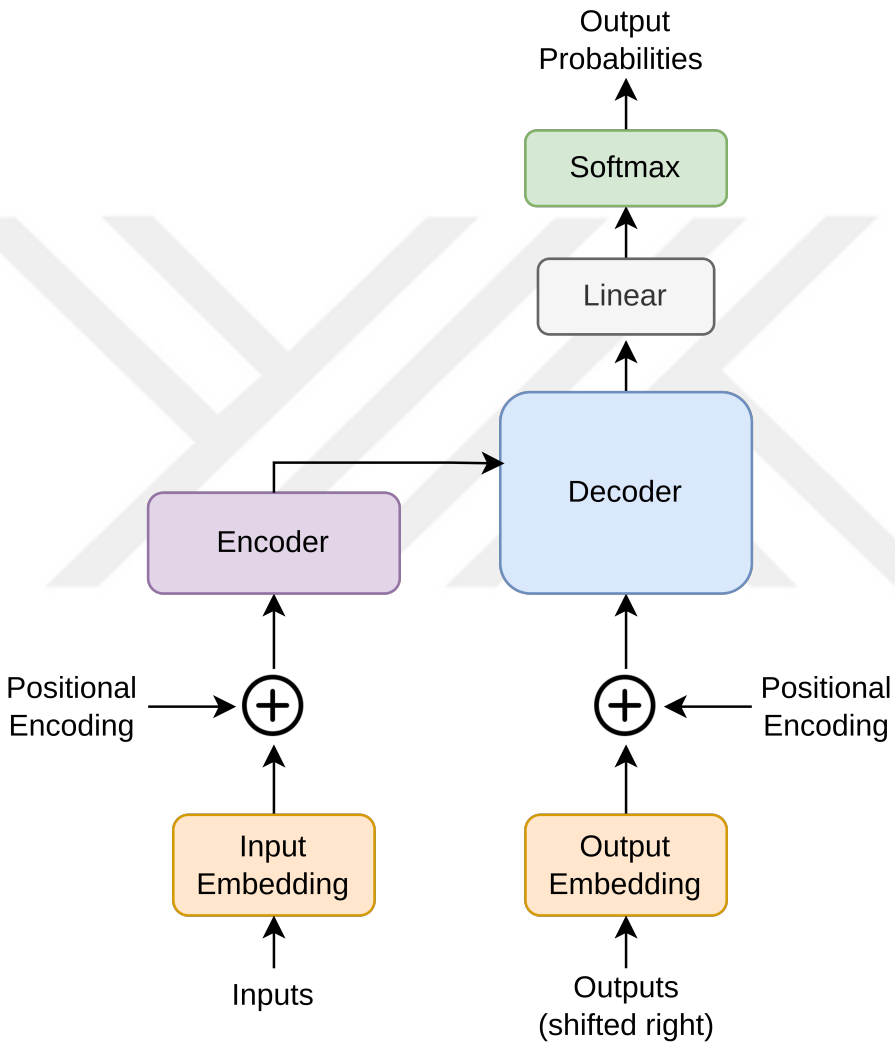

FIGURE 3.1: Model architecture of Transformer [44]

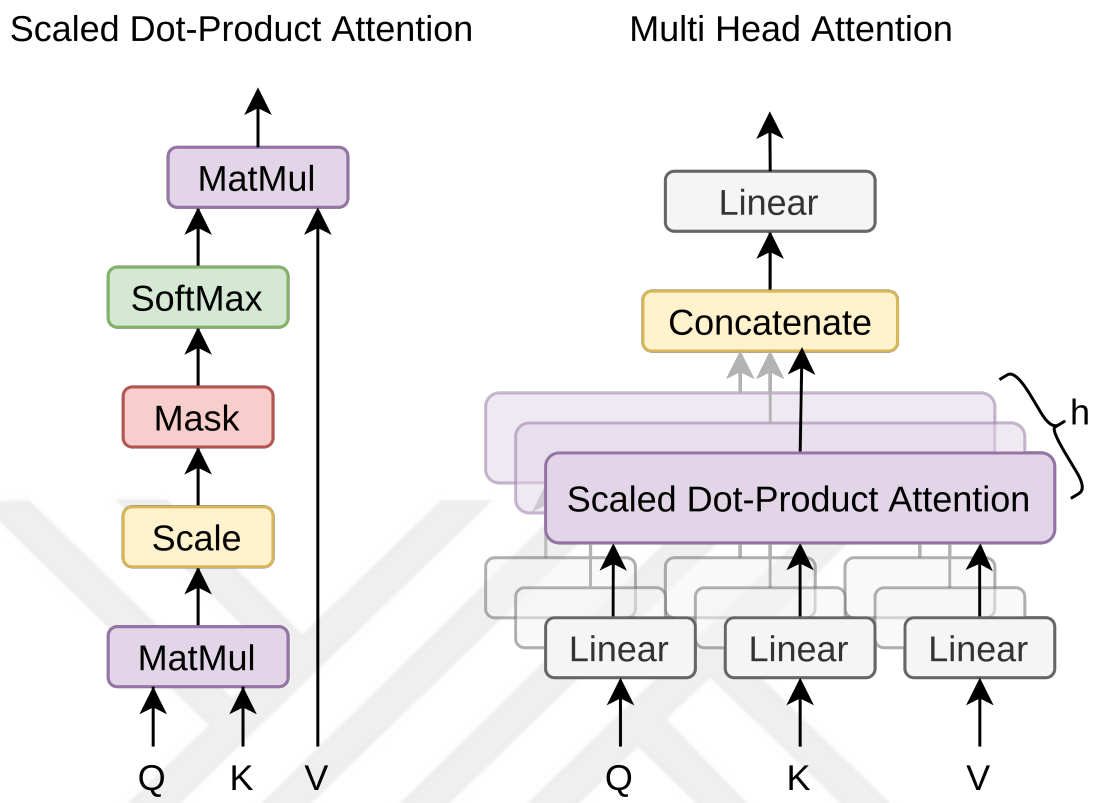

FIGURE 3.2: Scaled Dot-Product Attention and Multi-Head Attention [44]

3.1.2 Attention

The definition of the attention function is mapping a query and a set of key-value pairs to an output that is the sum of values weighted by the compatibility of the query and key, which corresponds to the query.

Scaled Dot-Product Attention

Queries, keys of dimension d_k, and values of dimension d_v are taken by scaled dot-product attention as input. The keys of dimension d_k are used for scale factor when computing dot-products of the query by dividing each key to $\sqrt{d_k}$. After that, softmax is applied to values for weighting. Scaled Dot-Product Attention layer is shown on the left in Figure 3.2.

The scaled dot-product attention is computed as follows:

$$Attention(Q, K, V) = \left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{3.1}$$

where Q is the query set matrix, K and V are for the keys and values respectively. K^T is the transpose of the K.

Multi-Head Attention

Each query, key, and value are projected to linear space h times by independent linear projections for d_q, d_k, and d_v dimensions. The attention function for these linearly projected queries, keys, and values is applied in the manner of parallel. The output of attention functions which is at head h, are concatenated with each other and another linear projection is applied over this concatenation. Multi-Head Attention layers are shown on the right in Figure 3.2.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots, head_h)W^O \tag{3.2}$$

where

$$head_h = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.3}$$

where weights of liner projections $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

3.1.3 Position-wise Feed-Forward Networks

There is a fully connected feed-forward network that is performed at each position separately and with the same weight for different positions. Each layer has different weights for the feed-forward network.

The Feed-forward network performs two linear transformations and it has the ReLU activation function which is in between linear transformations.

Fully connected feed-forward performed as follow:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.4}$$

3.1.4 Positional Encoding

Transformer has no information about the position of the tokens in the sequence by relatively or absolutely due to Transformer does not contain any recurrence or convolution. For the sake of giving positional information, there is the positional encoding for adding to input embeddings at the beginning of encoder and decoder stacks. To sum the positional encoding and input embeddings, they have the same dimension d_{model}.

The positional encoding obtained as given sine equation:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.5}$$

where positions of the token in the sequence are shown as pos and the dimension shown as i.

3.2 Bidirectional Encoder Representations from Transformers

BERT (Bidirectional Encoder Representations from Transformers) is a language representation model that is designed to pre-train on unlabeled text corpus by bidirectional to learn the context of the text. Transformer applies bidirectional self-attention, each token interacts with other tokens in BERT architecture.

The pre-train objective of the BERT is the masked language model (MLM). Randomly selected tokens in a text sequence are masked and the model tries to find masked words by using the context of the text extracted from unmasked terms in the task of the masked language model.

The language representation models are pre-trained by two known approaches which are *future-based* and *fine-tuning*. The unidirectional language models such as ELMo [28] and Generative Pre-trained Transformer (GPT) [29] have the same objective for pre-training. ELMo applied a future-based approach, it has task-specific features. GPT has fewer task-specific features and it can be trained for any other task by simply fine-tuning.

3.2.1 Model Architecture

The architecture of BERT consists of Transformer described by Vaswani et al. [44] in multi-layer bidirectional form. Transformer applies bidirectional self-attention, each token interacts with other tokens on both left and right sides in BERT architecture.

In BERT architecture, L stands for the number of layers, in other words, Transformer blocks, H stands for the hidden size, and A stands for the number of self-attention heads. Base BERT model has 12 layers, 768 hidden sizes, and 12 self-attention heads. The total number of parameters is 110 million for the base model of BERT.

3.2.2 Input/Output Representations

BERT has special tokens to represent single sentences and pairs of sentences like question-answer tasks without any ambiguation to create a generalized language model by independent from any task. Each text sequence starts with the special classification token $[CLS]$ which represents the final hidden state that aggregated through the sequence, and this token is used on the classification tasks. To separate sentences in a pair of text sequences, the special separator token $[SEP]$ is used between the first sentence and the second sentence.

BERT uses WordPiece embeddings [46] to tokenize text sequences into tokens of vocabulary which has 30.000 unique tokens. WordPiece tokenization handles decode single character and full word without suffering mismatch.

3.2.3 Pre-Training

BERT pre-trained two unsupervised tasks that are masked language model (MLM) and next sentence prediction (NSP). BERT was pre-trained with a large text corpus by using BooksCorpus which consists of 800M words and English Wikipedia which consists of 2.500M words.

Masked Language Model

In the masked language model, randomly 15% of tokens are masked of all tokens in a text sequence that tokenized by WordPiece. When pre-train BERT by MLM task, the model only predicts to masked words. With help of bidirectional

transformer architecture, each token interacts with any other tokens in both left or right ways. In this way, masked terms are predicted by using the context of the whole text sequence.

Randomly chosen tokens in a text sequence are replaced with the special token $[MASK]$ by 0.8 probability, by 0.1 probability a random token is used instead of the masked one, and the other 0.1 probability chosen token is used as it is without masking.

Next Sentence Prediction

Some NLP tasks like Question Answering are based on detecting interaction between two sentences. To detect the relationship between pairs of sentences, BERT pre-trained for the next sentence prediction (NSP). In half of the training data, the second sentences are actually the continuation of the first sentences. In the other half, the second sentences are unrelated to the first ones. BERT is trained to predict the first sentence is followed by the second sentence or not in the NSP task.

3.2.4 Fine-tuning

BERT pre-training creates a language model that can extract contextual embeddings with the help of Transformer's self-attention mechanism [44]. Through this, BERT can be used in many language processing tasks by only fine-tuning.

The pre-trained BERT model is used as a starting point to fine-tune a specific task. The model is fine-tuned end-to-end using task-specific inputs and outputs. The aggregated representation token $[CLS]$ is feed-forward into classification layers on classification tasks. At the end of fine-tuning, all parameters in BERT and task-specific layers are updated.

The fine-tuning requires less time compared to pre-training of BERT.

3.3 Text to Text Transfer Transformer

Raffel et al. [30] proposed a model that approaches each natural language processing (NLP) task as a text-to-text problem. The idea of the Text to Text Transfer Transformer (T5) is the learning of a language model by using a huge amount

of English text knowledge and fine-tune for downstream tasks which are question answering, document summarization, sentiment classification, and machine translation.

The rich dataset was obtained from the internet. Each month, nearly 20TB of text data is written on the internet. The web-based text data called "Colossal Clean Crawled Corpus", filtered by some preprocessing such as discarding web pages, which contain placeholder "lorem ipsum" text, slangy words, programming languages, or short web pages. The cleaned text corpus is used to learn a generative language model to become applicable for the text-to-text tasks.

3.3.1 Model Architecture

T5 tries to explore the limits of transfer learning by understanding the text and fine-tuning for tasks by learning a language model for the English language with a rich and clean text corpus. To this end, the T5 model uses a similar architecture with Transformer as proposed by Vaswani et al. [44] instead of proposing a new network architecture. To actualize the text-to-text approach, the generative language model is required, therefore, BERT [6] language model is not applicable for this task. Though, the configuration of the encoder and decoder of Transformer is the same with the $BERT_{BASE}$ model structure.

T5 model architecture has some differences from Transformer. One of them is removing layer normalization bias and adding to the residual path from outside as layer normalization. Another difference is relative position embeddings [37] are used instead of sinusoidal position embedding.

3.3.2 Input/Output Representations

Each downstream task is considered a text-to-text task. Therefore, a task-specific prefix is used before the input text as an indicator for the downstream task. For example, to translate an English sentence to German, the prefix will be "translate English to German:" and that is followed by the input sentence. And the output of this input sequence is the translation of the English sentence to the German one. Another example, to summarize a long text, the prefix "TL;DR:" (stand for "too long, didn't read") is added to the text. The input text is given with this prefix and the output is the summary of the input.

The input text sequences are represented with WordPiece [46] tokens as same as BERT input representation. T5 model fine-tuned to translate English to German, French, and Romanian, therefore, the WordPiece vocabulary extended to cover other languages. The web pages are used to crawl text in these languages and expand vocabulary. At the end of this, the T5 model only supports a constant set of languages.

3.3.3 Pre-Training

Pre-training is performed on the T5 to learn a generative language model before fine-tuning to apply downstream tasks. The pre-training was done in the manner of the unsupervised learning text-to-text task. T5 model pre-trained with cleaned web crawled text corpus (Colossal Clean Crawled Corpus) from April 2019. The dataset consists of about 750GB of text data.

T5 pre-training task is inspired by BERT's "masked language model". In T5, the masked language model is masking randomly chosen tokens in the input sequence and the T5 model tries to predict the original text. Additionally, prefix language modeling which is the next sentence prediction in BERT's pretraining task and deshuffling input text is evaluated by T5. The deshuffling language model is the put the text in order to original form from a scrambled input sequence.

In all pre-training objectives, the input and the output of the T5 are used as the text sequence. To measure the loss of the generative performance of the T5 model, the cross-entropy loss was used, and the model was optimized with AdaFactor [38]. When prediction, at each timestep highest probability is chosen, which is called greedy decoding.

3.3.4 Fine-Tuning

After pre-training with text-to-text tasks to learn a generative language model, T5 is fine-tuned for specific tasks which can be also solved as text-to-text objectives. All downstream tasks are combined into a single model by fine-tuning by adding a prefix to the input text sequence to specify the task.

The downstream tasks like question answering, document summarization, sentiment classification, and machine translation are fine-tuned by identifying with

task-specific prefix and the output of the model is the expected text of these tasks. For classification tasks, the expected output is the target class in text format. Thence, the multiple NLP tasks are handled with a single model.

4. SEARCH ENGINE

Search engines are tools that retrieve related information about user requests. Most of the time, user fill a form which has different fields such as text, date, or list, the search engines try to find relevant information that is in kind of document, web page, etc. by seeking the form of request. Search engines can be specialized for a specific area like library search tool or more general like web search tools that can search text or media.

A search engine has two main objectives. The first objective is retrieving relevant information for an input request, and the second one, ranking retrieved information by their relevance score according to the input request.

For this purpose, a search engine may have a single-stage or multi-stage. Single-stage search engines produce results in a single fetching and ranking algorithm. In multi-stage search engines, retrieving possible related information and ranking them are separated into two stages. Multi-stage searching provides using a lightweight information retrieval algorithm to select possible documents as a subset of the collection. After that, a complex ranking algorithm can be used to sort a subset instead of ranking the whole collection for each input request.

In multi-stage search engines for text data, first, the collection is ranked in an efficient way and creates a subset for the input query. After that, a more effective ranking algorithm re-rank the subset of the collection. To achieve easy access to documents that contain one of the words in the query, the inverted index method is used. To determine the most possible documents, retrieved documents are ranked by BM25.

In this thesis, the search engine specialized for searching text documents by text queries. The main focus of this work is ranking all documents in the collection. Improving first-stage retrieval performance by maintaining the efficiency of the first-stage retrieval on the search time is aimed.

4.1 Lucene

Lucene[1] is an open-source framework that provides text indexing and searching capabilities. Lucene framework is written in Java programming language. Lucene is a lightweight and reliable search engine specialized in text search. It requires a small amount of memory usage, and it works high performance. Lucene has fielded text search and it can rank documents according to queries with a similarity metric like BM25. Lucene supports more than 10 languages and some language-specific processing can be applied on the index time and query time.

4.1.1 Text Index

Lucene index documents with inverted index method. The inverted index is a way to access documents by searching a set of words without requiring investigating all documents in the corpus. The inverted index is a data structure that stores documents, and by searching words, can easily reach documents in the index. To create an inverted index, first, unique words are extracted from the corpus, and it is a vocabulary of the corpus. Unique words are used for constructing an index table. After that, document references are mapped to words if they contain the word in the index table. Thus, when a word is searched, the set of documents that have the word is accessed by $O(1)$ complexity.

In this thesis, the English language corpus is used to evaluate methods. Therefore, a language-specific analyzer, the English analyzer is used, Lucene has a built-in language analyzer for English. Lucene also provides a set of options that can be applied like ignoring stopwords or stemming words. These preprocessing are applied in search time too.

Default English analyzer applies pre-processes on the corpus consecutively which are word tokenizer, possessive filter, lower case filter, stopword filter, and porter stemmer in the index time.

- **Word Tokenizer:** A text corpus is tokenized into the words according to word break rules by using the Unicode Text Segmentation algorithm.
- **Possessive Filter:** The possessive filter gets rid of possessives (trailing 's) from words.

[1] Lucene - lucene.apache.org

- **Lower Case Filter:** All word tokens converted to lowercase.
- **Stopword Filter:** Removes words that are stopwords from the text.
- **Porter Stemmer:** Words are transformed to the stemmed version by porter stemmer algorithm. Porter stemmer normalizes terms by removing the commoner morphological and inflexional endings from words in English.

4.1.2 Query Search

The query search is done term by term. TF-IDF scores are calculated for each term in the query and cumulative scores of them give the document score for the query for BM25 similarity. The same pre-process in the index time is applied to the query before document search. Additionally, there are several options when searching a query like a phrase search and fuzzy search. There is a boost factor for query term weighting as well.

With the help of query term boosting, query term weighting methods can be easily evaluated by adding weight to terms to query before search. With reference to this, there can be a coefficient for each term as a boost that is multiplied by the term score to define the importance of the term.

```
what^0.137 is^1.043 dynamic^0.812 resolution^0.910
```

Lucene query search provides phrase search instead of word by word document search. The sequential words can be searched as a single term and the phrase is searched in the document in the same way. At this stage, with fuzzy search, phrases match with phrases in the documents if the edit distance is less than the defined fuzzy value.

```
what is "operating system misconfiguration"~3
```

5. RELEVANCE FEEDBACK

The ranking is the task of finding relatively higher scores for relevant documents than irrelevant ones. In this way, relevant documents are placed at the top of the output result list. Relevance feedback information identifies if the presented documents are relevant or not. While this information is not usually available, methods using pseudo-relevance feedback exists that utilize user interactions to adjust the retrieval methods. Term recall is such a method that uses the term weights to adjust the retrieval function like BM25 to retrieve relevant documents.

BM25 score of a query Q that consists of terms $q_1, q_2, \cdots, q_n$, and document D, is calculated as follows [43]:

$$BM25(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot TF(D,q_i) \tag{5.1}$$

$$TF(D,q_i) = \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)} \tag{5.2}$$

where $f(q_i, D)$ is term frequency of q_i in the document D, $|D|$ is length of the document. Average document length is given as $avgdl$. k_1 is the parameter to control the rate of term-frequency saturation and b is the parameter for the effect of document length normalization. $IDF(q_i)$ is the inverse document frequency that contain q_i is shown as:

$$IDF(q_i) = \log\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right) \tag{5.3}$$

where N is the number of documents and $n(q_i)$ stands for number of document that contains qi.

BM25 equation with weighted terms can be written as follow:

$$BM25_{tw}(D,Q) = \sum_{i=1}^{n} w_i \cdot IDF(q_i) \cdot TF(D,q_i) \tag{5.4}$$

where w_i is weight of ith term of the query.

When ranking documents for the given query, each term has a different effect on fetching relevant documents. Some terms are more meaningful to understand the keywords of the query while others serve a more grammatical purpose. Term recall is a probabilistic weight function for setting w_i weights. It uses only relevant documents to weighting the terms, and without using the information in other irrelevant documents in the corpus.

5.1 Term Recall Based Term Weighting

The importance of the query term in the given query to retrieval success is determined by the term recall value. Term recall is the ratio of the number of relevant instances that contain the term over the total number of all relevant instances. Thus, term recall shows the usage frequency of a term in the relevant instances and contribution to fetch relevant ones.

Term recall can be applied over two side of term weighting. For query term weighting, term recall is number of relevant documents with the term divided by number of all relevant documents, and the equation is as follows:

$$TermRecall(t, q) = \frac{|D_{q,t}|}{|D_q|} \tag{5.5}$$

where $D_{q,t}$ is set of relevant documents that contain the term of query and D_q is set of relevant documents for the query. While this weighting scheme rewards terms appearing in relevant documents, it does not penalize terms with low discriminative value. A stop word appearing in all documents both relevant and irrelevant will receive the maximum weight under this framework.

For document term weighting, this time, the number of queries that address to document is used. The term recall value of a query term is the number of queries that contain the term for the document divided by the number of all queries that address the document. Term recall for document term weighting is shown as follows:

$$TermRecall(t, d) = \frac{|Q_{d,t}|}{|Q_d|} \tag{5.6}$$

where Q_d is the set of queries for document d and $Q_{d,t}$ is the subset of Q_d that contains term t. Due to the queries being shorter than the documents, for all document terms, a term weight cannot be produced under the term recall scheme.

5.2 Pairwise Ranking Loss Based Term Weighting

We propose an optimization function that learns optimal weights for query terms to achieve a higher weighted BM25 score for relevant documents than irrelevant ones.

Let $BM25(D, q_i)$ be BM25 value of the ith term of the query for document D. In this case, BM25 score of document D with weighted terms equals to $w_1 \cdot BM25(D, q_1) + w_2 \cdot BM25(D, q_2) + \cdots + w_n \cdot BM25(D, q_n)$. In this equation, n is the number of terms in the query, and it does not change for different documents. For this equation, we try to optimize term weights to obtain a higher final BM25 score for relevant documents and bring forward the most important query terms for retrieval performance.

Sorting is based on pairwise comparisons between the documents. The documents which are relevant to the query should have a higher rank compared to the irrelevant documents. To find term weights that lift the score of relevant documents, an optimization based on pairwise ranking loss is designed.

For an input query $Q = \{q_1, q_2, \cdots, q_n\}$, relevant documents are paired with irrelevant ones to form a training instance. A training instance is represented with two feature vectors formed of BM25 values for each term in the same order with query terms. Feature vectors can be represented as $D_{Rel} =< x_1, x_2, \cdots, x_n >$ for relevant document and $D_{Irrel} =< y_1, y_2, \cdots, y_n >$ for irrelevant document in the pair. BM25 score of the relevant document is written like $BM25_{tw}(D_{Rel}, Q) = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n$. In the same way, for irrelevant document BM25, score is $BM25_{tw}(D_{Irrel}, Q) = w_1 \cdot y_1 + w_2 \cdot y_2 + \cdots + w_n \cdot y_n$.

The final objective is to learn term weights that produce high scores for relevant documents while reducing the score of irrelevant ones. The pairwise optimization aims to provide the following condition, and the loss function can be defined as the next one:

$$BM25_{tw}(D_{Rel}, Q) \geq BM25_{tw}(D_{Irrel}, Q) + \alpha \tag{5.7}$$

$$loss = \frac{[maximum(BM25_{tw}(D_{Irrel}, Q) - BM25_{tw}(D_{Rel}, Q) + \alpha, 0)]^2}{2} \tag{5.8}$$

where α is the margin between relevant and irrelevant document pairs, assuring that the difference between a relevant and irrelevant document is at least α [35]. If the weighted BM25 score of the relevant document is greater than sum of the margin and the irrelevant one, then the loss will be 0, and weights are not updated.

Let $BM25_{tw}(D_{Rel}, Q)$ be equal to $\vec{w} \cdot \vec{x}$ and $BM25_{tw}(D_{Irrel}, Q)$ equals to $\vec{w} \cdot \vec{y}$, and $\cdot$ is dot product. If relevant score is less than the irrelevant one and margin α, $\vec{w} \cdot \vec{x} < \vec{w} \cdot \vec{y} + \alpha$, derivative of loss with respect to weights as follows:

$$\frac{\delta loss}{\delta \vec{w}} = (\vec{w} \cdot \vec{y} - \vec{w} \cdot \vec{x} + \alpha)(\vec{y} - \vec{x}) \tag{5.9}$$

To find optimal weights using a gradient, Adam [16] optimizer is used. Adam optimizer is an optimization algorithm of stochastic objective by based gradient on first-order. For each weight, individual adaptive learning rates are used in Adam optimizer. Rescaling of the gradient does not vary parameter changing values and hyperparameters limit the step-sizes. The weights are updated at each iteration to find the optimal solutions by the following equations.

Biased first moment and second raw moment estimation updated as follow:

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{5.10}$$

$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{5.11}$$

where β_1 and β_2 is moment estimation parameters as exponential decay rates in range $[0, 1)$, and the initial values of the first and second moment vectors are 0.

Bias corrected first and second raw moments are calculated as follow:

$$\widehat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right) \tag{5.12}$$

$$\widehat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right) \tag{5.13}$$

where t is the timestep, in another word, the number of iteration. Timestep t increased by 1 at the beginning of each iteration and the initial value is 0.

Finally, the weights are updated as follow:

$$w_t \leftarrow w_{t-1} - \alpha \cdot \widehat{m}_t / \left(\sqrt{\widehat{v}_t} + \epsilon\right) \tag{5.14}$$

where α is the stepsize and ϵ is the very small number to prevent undefinition due to zero division in the equation.

Initial weights are defined as a normal distribution with 0.5 mean and 0.05 standard deviation. After pairwise ranking loss-based term weighting, some of the optimal weights for query terms can be negative. That means the optimal solution is provided by penalizing unwanted terms. The negative term weights are not applicable by search framework Lucene. Therefore, all weights must be greater or equals to 0. To provide this constraint, there are two ways, the weights are normalized after optimization or force the weights to be positive when searching optimal solution.

The first way is the $min - max$ normalization, the optimized term weights fit the range 0 and 1. When learning optimal term weights, we can force the weights to be positive with constraint. For this purpose, a cost value is added to the loss for being negative weights, or another solution, negative values are projected to zeros in each iteration after updating weights.

5.2.1 Min-Max Normalization

Final weights, optimized for the higher BM25 score for relevant documents, are in the range $-\infty$ and ∞. After the optimization, to scale the weights between 0 and 1, min-max normalization applied as follows:

$$Scaled\ w_i = \frac{w_i - W_{min}}{W_{max} - W_{min}} \tag{5.15}$$

where w_i is the ith term weight for query, W_{max} is the maximum term weight for the query, and W_{min} is the minimum term weight for the query.

5.2.2 Minimization Absolute Values of Negative Weights

Positive term weights let us be sure that optimized weights promote the most valuable terms in the query to fetch relevant documents. However, negative weights refer to reduce document score by penalizing unwanted terms.

To prevent this, a cost function that aims to minimize absolute values of the negative weights is added to the loss. Irrelevant documents cause negative weights due to the loss, to this end, the additional cost function tries to minimize the gap between negative scores of terms and zero. Negative scores are calculated by element-wise multiplication of estimated weights and feature vectors of the irrelevant documents. The document features are always positive due to BM25 calculation and negative scores are caused by negative weights.

For this objective, the additional cost function written as follow:

$$cost = |minimum(\vec{w} \odot \vec{y}, 0)| \tag{5.16}$$

where $\odot$ is element-wise multiplication, $\vec{w}$ is weights and $\vec{y}$ is feature vector of irrelevant features.

The derivative of the cost function by w is:

$$\frac{\delta cost}{\delta \vec{w}} = sign((\vec{w} \odot \vec{y}) \odot \vec{y}) \tag{5.17}$$

when $\vec{w} \odot \vec{y} < 0$ otherwise 0.

At the end of the optimization, there can be still negative weights even if they are close to 0. To prune negative weights, the weights below zero are ignored and projected to 0 as following equation:

$$\vec{w} = maximum(\vec{w}, 0) \tag{5.18}$$

5.2.3 Non-Negative Weight Constraint

In the training time, the negative weights are projected to another space by assigning them to 0 at each iteration after updating weights. In the next iteration, the equation tries to find another solution without negative weights. If a weight

tends to negative values, it is prohibited by ignoring weights less than 0 at each iteration. The final solution is in positive space with the help of the non-negative constraint.

In each iteration, the following operation is applied to optimized weights after updating weights by gradient:

$$\vec{w}_{t+1} = maximum(\vec{w}_t, 0) \tag{5.19}$$

where $\vec{w}_t$ is the updated weights by optimizer and $\vec{w}_{t+1}$ the new weights for next iteration. The non-negative constraint is like $ReLU$ activation function, but it is applied to the weights instead of the output.

5.2.4 Pair Selection

Relevant and irrelevant pair selection starts with the initial BM25 ranking without weighting terms. Relevant instances are known for a query thanks to the labels, and they are picked from the collection. Ranking documents by original (unweighted) queries is called the initial run. Irrelevant instances are selected from the top results that are retrieved by the initial run, after excluding relevant instances for the query. This allows the model to discriminate the relevant documents from irrelevant false positives of the default BM25.

Each relevant document is matched with irrelevant documents from the initial run to construct pairs. Pair generation is based on cross-production of the relevant and irrelevant documents. The number of pairs at the end of the pair generation depends on the number of documents retrieved by the initial run and the number of relevant documents. Most of the queries fetch the desired number of documents from the initial ranking, but some of them may be less result if there is less match with the documents.

Each query is optimized independently from the other queries. This method in a way overfits weights with respect to relevant and irrelevant documents per each query. In this way, the optimization model predicts the best weights to increase relevant document weights and decrease irrelevant ones when compared to each other. It should be noted that as these weights would be too specific to a query and not generalize to a different query, they would not be useful for a search

engine. However, as we proceed to predict these weights for an unseen query using contextualized word embeddings, considering both the semantics of the query and the role of the term in the query, it is applicable to unseen queries. After estimating optimal weights, a contextual model like BERT learns the predicted weights by only using the query terms.

In short, the pairwise term weight optimization method learns specific term weight for each query by overfitting relevant and irrelevant pairs for the query. A language model like BERT aims to understand contextual information of queries to predict these optimized term weights by learning interactions between terms in the query. In the end, a term weighting model is trained that estimates pairwise term weight for each term in a query.

TABLE 5.1: Relevant and irrelevant pair for a query.

Query	What is dynamic resolution?
Relevant	DynamiX is a unique implementation of real-time **dynamic resolution** technique that is designed to enable a tunable minimum performance level to increase the playability of a game by **dynamically** changing the render target **resolution** of objects in real time, without the need of the game developer to design it in advance.
Irrelevant	Imaging the larynx's positions and the vocal folds' vibrations is possible using **dynamic** MRI. This technique permits measurements of laryngeal structures and glottal parameters in **dynamic** function with multiplanar high-**resolution** imaging.

*Query and passages were taken from MS MARCO passage dataset.

6. TERM WEIGHTING

The term weighting adds more contextual knowledge to text than statistical information from term frequency and inverse document frequency [5, 17, 49]. For specialized tasks, term weights are estimated with the help of contextual knowledge. The specialized task is determined as document retrieval and ranking. To this end, the term weighting can be applied to the documents in index time or to the queries in the search time. By this means, term weighting carries much more retrieval and ranking information than the classic TF-IDF features.

In this thesis, document and query term weighting methods in the literature were examined. The BERT-based term weight estimation model was evaluated with the proposed pairwise relevance feedback. In the following sections, the query term weighting methods in literature and proposed BERT-based estimation model are given. Next, the document term weighting methods in the literature are explained.

6.1 Query Term Weighting

Query terms are weighted to indicate how essential their contribution to the query for retrieval and ranking documents. Contextual knowledge is used for query term weighting. The documents are retrieved and ranked with matched query terms. The cumulative scores of the query terms give the document relevance score for the query. As mentioned in Chapter 5, Relevance Feedback, predicted weights are used as a coefficient.

DeepTR and DeepCT-Query use term recall as relevance feedback. To predict target relevance feedback, DeepTR uses a term-query difference vector calculated by word embedding vectors. DeepCT-Query is the same framework as DeepCT-Index, DeepCT-Query handles term weighting on the query side. Additively, a proposed query term estimation model based on BERT is explained in this section.

6.1.1 DeepTR

DeepTR [49] is a method that weighting query terms by their distance to query itself. The distance between term and query is measured by word embedding

vectors. The word embedding is weighting words to represent according to their semantic values as a k dimensional vector. Embedding vector of word w_i in k dimensional space is shown like $w_i = \langle x_0, x_1, x_2 \cdots x_k \rangle$. Word2Vec [19] word representation as vector space method was used in DeepTR.

The importance of the query term in the given query to retrieval success is determined by the term recall value. Term recall for query term weighting is a ratio of the number of relevant documents that contain the term over the total number of all relevant documents. Thus, term recall shows the usage frequency of the query term in the relevant documents and contribution to fetch relevant ones.

Features

The features for each term in a query are extracted by its distance to the query. The distance is calculated by word embedding vectors. A query is represented in a k dimensional space by taking average word vectors of terms that form the query and the feature of a term is the vectorial distance between the word embedding vector of the term and the query.

Let a query is represented as $q_i = t_{i1}, t_{i2}, t_{i3}, \ldots, t_{in}$, and w_{ij} is a word embedding vector for t_{ij}. In this case, the feature vector x_{ij} of t_{ij} calculated as follows:

$$x_{ij} = w_{ij} - \bar{w}_{q_i} \tag{6.1}$$

where

$$\bar{w}_{q_i} = \frac{1}{n_i} \sum_{k=1}^{n_i} w_{ik} \tag{6.2}$$

Training

DeepTR model tries to estimate the term recall value for each query term. The input of the model is the distance of the term to query. The target values of the model are *logit* of term recalls. The target value for term t_{ij} is calculated as follows:

$$y_{ij} = log \frac{r_{ij}}{1 - r_{ij}} \tag{6.3}$$

where r_{ij} is term recall value of the term t_{ij}.

The model tries to learn weights of β in the equation of $y = \beta^\top x$. Before the optimization l_1-norm regularization applied to input features. Finally, the model tries to optimize LASSO regression as follows:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{n_i} \left(y_{ij} - \beta^\top x_{ij} \right)^2 + \lambda \|\beta\|_1 \tag{6.4}$$

where M is number of queries, n_i is term number of ith query, x_{ij} is feature vector and y_{ij} target of ith query's jth term.

At the end of the model estimation, sigmoid values are computed for predicted weights to map predictions in $[0, 1]$ range. Final query term weights are calculated as follows:

$$\widehat{\mathbb{P}(t \mid R)} = sigmoid\left(\hat{\beta}^\top x\right) = \frac{\exp\left(\hat{\beta}^\top x\right)}{1 + \exp\left(\hat{\beta}^\top x\right)} \tag{6.5}$$

6.1.2 DeepCT-Query

DeepCT [5] is a framework to weighting terms based on contextual embeddings. Term frequency does not contain any clue about the context of the text, and it can be equal for relevant and irrelevant documents. To extract contextualized embeddings for words in the text, the BERT [6] natural language model is used.

DeepCT is a regression model that tries to learn target term weight value by contextual knowledge of the text for each word in the text by fine-tuning the BERT model. The input of the regression layer of the model is the tth contextual embedding of the BERT sequential output. Word importance weight is calculated by DeepCT as follow:

$$\hat{y}_{t,c} = \vec{w} T_{t,c} + b \tag{6.6}$$

where $T_{t,c}$ is the contextual embedding for text c, w is the weight and b is the bias of the linear regression.

DeepCT tries to minimize mean squared error (MSE) between target and predicted weights.

$$loss_{MSE} = \sum_{c} \sum_{t} (y_{t,c} - \hat{y}_{y,c})^2 \tag{6.7}$$

where y is the target weights and $\hat{y}$ is the predicted weights.

Term weight predictions of DeepCT are in the range between $-\infty$ and ∞, still, most predictions remain in the $[0, 1]$ range due to ground truth weights in the $[0, 1]$.

BERT tokenization generates subwords for words that are not in the vocabulary of BERT. When computing MSE loss, the first subword is used for evaluating as the entire word and other subwords are masked in DeepCT. For example, DeepCT is tokenized as *"deep"* and *"##ct"*, the loss of the model prediction is computed over *"deep"* which is the first subword and *"##ct"* is masked out.

DeepCT-Query [5] is a way to the usage of DeepCT framework for weighting query terms. In DeepCT-Query, the terms of the queries are weighted and the term recall value is used as a relevance feedback target. The similar way with DeepTR, target term weights of queries are defined by the term recall for query term weighting.

In the search time, estimated query term weights are multiplied by $BM25$ relevance score of the term to boost score by term importance, as mentioned in Chapter 5, Relevance Feedback.

6.1.3 Term Weight Prediction Model

Pairwise term weight optimization model runs in the manner of supervised learning. The model must know relevant and irrelevant pairs for the query with the query itself. For this reason, weight estimation cannot be made for new queries in the search-time. A BERT based regression model is trained with the weights optimized pairwise, the BERT model uses only the input query to predict term weights of the query.

To fine-tune the BERT model, a regression task targeting query term weights is used. BERT takes term and query as a pair. The term of the query and query

itself are separated with special token $[SEP]$. This way, the interaction between individual terms and queries is learned. The pooled output of BERT which is $[CLS]$ token vector is followed by a fully connected feed-forward layer with a single output value. The output layer has no activation function due to trained as a regression task. To prevent overfitting and achieve more generalization there is a dropout layer with 0.2 probability between pooled output and fully connected layer. The model is trained with the MSE (mean squared error) by fine-tuning the pre-trained BERT model to estimates the term weights of the input query. The model is optimized with Adam [16] optimizer and 10% of training steps are used as warm-up steps.

All training pairs have a target value between 0 and 1. For this reason, most of the predicted term weights fall into this range. In the end, negative weights are ignored and set as 0. Non-negative term weights enable the use of boosting factors available in most search engines.

Thanks to interaction architecture, phrases which are sequential words can be weighted as terms. The first part of the input is the phrase in the query and the second part is the query, both of them are separated by the special token. The target weight predicted by the model is for a phrase instead of a single word in this case.

In search-time, the model estimates weights of each term of the query simultaneously. For each term in the query, model estimates weights as the coefficient for terms. An example for tokenization of the query and forming input given with BERT term weight learning model architecture at Figure 6.1.

The proposed term weighting model and DeepCT [5] framework use the same pre-trained BERT model to understand the context of the text to predict target term weight. The main difference between DeepCT and ours is the proposed model can estimate term weights for phrases with the help of the interaction architecture. DeepCT takes a query as an input and predicts term weights for each token in the query. In other respect, the proposed BERT-based term weight estimation model takes term and query as a pair, and it can predict term weights for tokens or phrases in the same architecture.

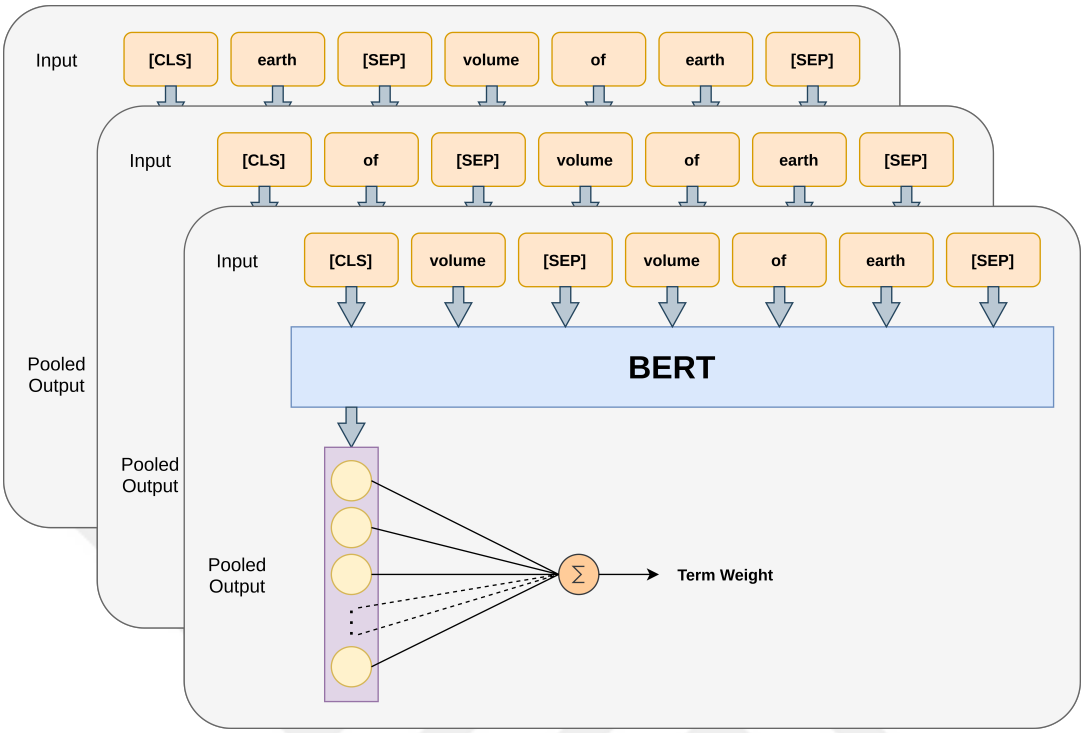

FIGURE 6.1: An example input and model architecture for term weight estimation.

6.2 Document Term Weighting

The main idea behind the document term weighting is rearranging term frequencies by repeating terms up to estimated term weights. Document term weighting can be investigated in two approximations. The first one is the use of existing terms to rearrange term frequencies. By the repetition of the terms by the estimated weights, term frequencies of the documents are formed up to represent contextual knowledge to increase retrieval and ranking performance.

The second approximation uses contextual knowledge to estimate possible queries that address this document. The document is expanded by predicted possible queries for itself. Thus, term frequencies that are important for retrieval and ranking, are increased. Additionally, if there are new terms in the document expansion, they reduce term mismatching between the document and the query in search time.

DeepCT-Index and TextRank methods based on term frequency increasing of existing terms. DeepCT-Index uses contextual information extracted by the BERT

language model, and TextRank extracts semantic relation between terms by using co-occurrences. Doc2Query and DocTTTTTQuery models use sequence-to-sequence language models that generate possibles query for the input document and predicted queries expand the document. Thus, term frequencies are changed and new terms are added to documents.

6.2.1 DeepCT - Index

The primary usage of DeepCT framework is the document term weighting. Both DeepCT-Query and DeepCT-Index tasks use the same framework on different sides of the term weighting and different target values. Target term weights for document term weighting task is defined by the importance of a term for the document. The importance of a term is measured by how necessary when fetching the document. For this purpose, term recall value is used as a metric for target term importance, for document term weighting, term recall calculates as the number of queries with the term for the document divided by the number of all queries that address the document.

The predicted term weights are converted to term frequencies by multiplying each term in the document by predicted weights. Most of the predicted weights are in the $[0-1]$ range, to multiplying terms, integer weights are required. For this reason, TF_{DeepCT} is a way of representing term weights as an integer value:

$$TF_{DeepCT}(t,d) = round(\hat{y}_{t,d} \times N) \tag{6.8}$$

where $\hat{y}_{t,d}$ term weight that predicted for term t in document d. N is a scale factor for prediction into the integer range. In DeepCT, $N = 100$ is used that provides two-digit precision is enough for this task.

An alternate way to make small values more visible is taking the square root of the predicted term weights and it causes the document to consist of more terms:

$$TF_{DeepCT}(t,d) = round(\sqrt{\hat{y}_{t,d}} \times N) \tag{6.9}$$

where $\hat{y}_{t,d}$ term weight that predicted for term t in document d with same as previous integer representation, and the scale factor N equals to 100 for square root representation in two-digit precision.

6.2.2 TextRank

TextRank [18] is a graph-based ranking algorithm. In fact, TextRank is used to extract keywords that important for the semantic of the text from a text. In a graph-based model, vertices represent words in the text and edges show the co-occurrence of words that linked. The base idea of the graph-based ranking algorithm is the showing importance of a word for the text by looking at the co-occurrence link for the word and the word's connections. As a projected result, words with more links and their connections have much more effect on the text.

PageRank [1] algorithm is used for word weighting by TextRank. Words in the text are represented as vertices of the graph and numbers of co-occurrence of words that formed together in a window-size range construct the weighted edges of the graph.

PageRank algorithm optimized as follows, when graph $G = (V, E)$ is a directed graph with a set of vertices V and edges E:

$$S\left(V_i\right) = (1-d) + d * \sum_{j \in In(V_i)} \frac{1}{\left|Out\left(V_j\right)\right|} S\left(V_j\right) \tag{6.10}$$

where $In(V_i)$ is the set of vertices that point to V_i and $Out(V_j)$ is the set of vertices that are pointed by V_i and d is a damping factor that can be assigned in range [0, 1] for fusing jumping probability from a peak to another in the graph.

TextRank can be used for weighting document terms. For each document in the corpus, TextRank algorithm applied to documents. Extracted keywords are weighted by TextRank confidence score. In a similar way with DeepCT, document terms are repeated in the document according to confidence scores. In this way, obtained term frequencies are used as term weighting.

6.2.3 Doc2Query

Doc2Query [24] uses a sequence-to-sequence transformer [44] model that predicts possible queries to be searched for a document. The document expansion process of Doc2Query model is given in Figure 6.2.

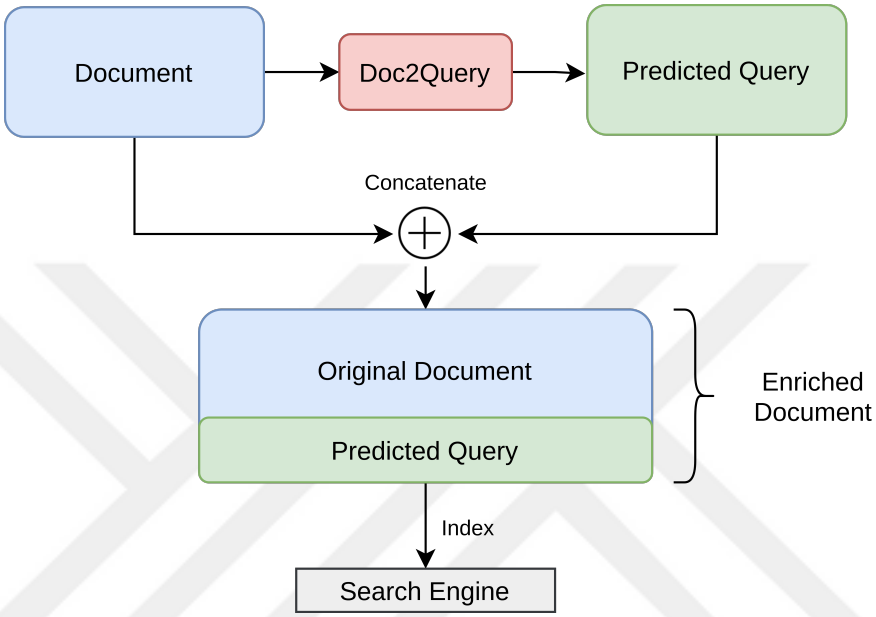

FIGURE 6.2: Doc2Query document expansion method [24].

Doc2Query transforms documents into queries with the help of the sequence-to-sequence transformer model. The model is trained with query-document pairs that are relevant to each other. After that, possible queries are predicted by the transformer model for each document in the corpus. Finally, generated queries are added to the end of the original documents. In this way, document terms weighting is provided by increasing the term frequency of possible words and adding new words to the document that helps to reduce vocabulary mismatching.

6.2.4 DocTTTTTQuery

DocTTTTTQuery [23] applies the same method of Doc2Query [24] that weighting terms of documents and expanding it with new words. In DocTTTTTQuery, T5 [30] is used instead of sequence-to-sequence transformer. For the rest, the model is fine-tuned with query-document pairs to predict possible queries that answer to relevant documents, in the same way with Doc2Query. Then, predicted queries

are added to documents for indexing as in Doc2Query. By the success of the T5 over the sequence-to-sequence model, better document terms weighting and better word estimating (to prevent vocabulary mismatching) are achieved.

7. EVALUATION

The retrieval and ranking performances are evaluated with different metrics and different cuts of the results. Performances of existing and proposed methods are evaluated with Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), Recall, and Normalized Discounted Cumulative Gain (NDCG) metrics. These evaluation metrics measure models on the performances of finding relevant documents and ranking correct order according to their relevance by input query.

The MAP measures the performance of getting relevant documents at the very beginning of the results. MRR and NDCG measure the ranking performance of the models based on the ranking positions of the relevant documents with the input query. The recall metric measures the performance of catching relevant documents in the top results. With these evaluation metrics, the performance of the models is compared by their retrieval and ranking of relevant documents to the input query.

With the help of the given metrics, the retrieval and ranking performances are evaluated. Additionally, the significance of the improvement by proposed methods to baselines was measured. The significance test was applied over evaluation queries for each metric by the paired t-test.

7.1 Recall

Recall shows the success of fetching relevant documents from all relevant ones. The recall at K is the ratio of the number of relevant documents in the top K results to the total number of relevant documents. The equation of recall is shown as follow:

$$Recall_K = \frac{|D_K \cap D_{Relevance}|}{|D_{Relevance}|} \tag{7.1}$$

where D_K is top K documents, and $D_{Relevance}$ is relevant documents with the input query.

7.2 Mean Average Precision

Precision shows the success of being relevant documents in the top results. The precision at K is the ratio of the number of relevant documents in the top K results to the number of ranked K top documents. The equation of precision is shown as follow:

$$Precision_K = \frac{|D_K \cap D_{Relevance}|}{K} \tag{7.2}$$

where D_K is top K documents, and $D_{Relevance}$ is relevant documents with the input query.

The **M**ean **A**verage **P**recision (MAP) is the average precision at K of all input queries. MAP is the division of the sum of precisions P_K for all queries to the number of queries, as follow:

$$MAP = \frac{\sum_{q=1}^{Q} P_K(q)}{|Q|} \tag{7.3}$$

where P_K is the precision of the query q at K and Q is the set of queries.

7.3 Mean Reciprocal Rank

The **M**ean **R**eciprocal **R**ank (MRR) is the average of the reciprocal ranks. Reciprocal rank is the inverse of the first relevant document index for a query. Reciprocal rank is calculated by dividing 1 by to rank of the document, and MRR is the sum of the reciprocal ranks of the query set divided by the number of queries, as follow:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{7.4}$$

where Q is the set of queries and $rank_i$ is the rank position of the first relevant document for the i'th query.

7.4 Normalized Discounted Cumulative Gain

Normalized **D**iscounted **C**umulative **G**ain (NDCG) [13] shows the success of the list order of documents according to their relevance score. $NDCG_K$ at K is the

discounted cumulative gain (DCG_K) that normalized by inverse discounted cumulative gain ($IDCG_K$).

Discounted cumulative gain at K (DCG_K) is a metric that penalizes the lower ranks for relevant documents. DCG_K is the sum of relation indicators that are divided by the logarithm of the rank indices. Relation indicator rel_i shows relevant or not of $i'th$ document with the input query.

$$DCG_K = \sum_{i=1}^{K} \frac{rel_i}{\log_2(i+1)} \tag{7.5}$$

The DCG_K depends on the query. When comparing more than one query, the normalization factor is required. The DCG_K is normalized across queries as follow:

$$IDCG_K = \sum_{i=1}^{|REL_K|} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{7.6}$$

where REL_K is the ordered list of documents that are relevant to the query by their relevance, and the size of the list is K.

For a query, $NDCG_K$ is computed as follows:

$$NDCG_K = \frac{DCG_K}{IDCG_K} \tag{7.7}$$

7.5 Paired t-Test

Paired t-test is a statistical hypothesis test that investigates the difference between two lists of data for the same subject. The similarity of the distribution between paired lists of data is tested with the help of paired t-test.

The equation of the paired t-test as follow:

$$t = \frac{\bar{d} - \mu_d}{s_d / \sqrt{n}} \tag{7.8}$$

where $\bar{d}$ is the sample mean difference, s_d is the standard deviation of the differences and n is the number of samples. The $H_0 : \mu_d = 0$ when it is null hypothesis and $H_1 : \mu_d \neq 0$ when it is alternative hypothesis.

The statistical significance threshold is defined as 0.05 to measure significance. When the calculated $p - value$ is less than the significance threshold, the alternative hypothesis is favored by rejecting the null hypothesis. In this case, there is a significant difference in the distribution of the results.

8. EXPERIMENTS

8.1 Dataset

In order to construct a framework that searches text queries in text corpus as a search engine, query-document pairs that are relevant to each other are required. Large corpus, queries which are for training and testing and their relation that indicate query-document pair is relevant or not, are essential to evaluate models with different approaches for effectiveness and efficiency of the search engine framework.

To meet these needs, the MS MARCO[1] [21] dataset that is widely used in related works for information retrieval and learning-to-rank tasks are used. The MS MARCO has two tasks which are document and passage ranking. The competition can be evaluated by ranking or re-ranking for both tasks. Document and passage ranking tasks use a large dataset that consists of the same queries and human-supervised labels. Labels are made that indicate relevance between queries and documents by human supervise.

To evaluate the usage of the neural ranking algorithms, the dataset of the MS MARCO passage retrieval task is chosen in this thesis. The passage ranking dataset has nearly 8.9 million passages, more than 5 hundred thousand training, and about 7 thousand evaluation queries. The exact number of records in the MS MARCO dataset is given in Table 8.1. The collection of the passages consists of a unique identifier *pid* for each passage, and the passage text. Train and evaluation queries have unique identifier *qid* and query text. The relations between query and passage are given as $pid - qid$ pairs.

[1] MS MARCO - microsoft.github.io/msmarco

TABLE 8.1: The number of passages and queries in MS MARCO collection.

Collection	Number of Records
Passages	8841823
Train Queries	502939
Dev Queries	6980

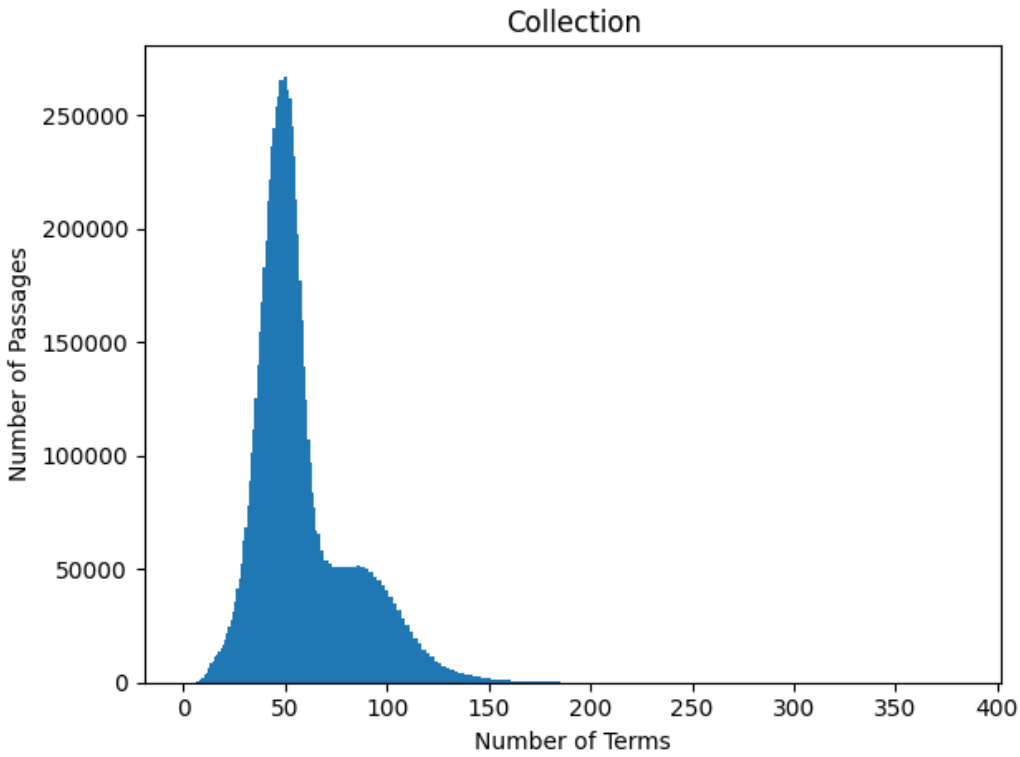

FIGURE 8.1: Histogram of the number of terms over passages.

The dataset consists of three-part. The passage collection was indexed, and all evaluations for the query term weighting task were run over this index. Document term weighting approaches manipulate original passage collection and the changed collection was re-indexed. The train queries were used in model training and parameter tuning. Parameters of BM25, k_1 and b, were tuned by randomly selected queries from train queries. In model training, part of training queries were selected as the validation set for the model evaluation. Dev queries in MS MARCO, were used as a test set, all evaluations and comparisons were performed with dev queries. The dev queries of MS MARCO are used as the test set for evaluated methods in the literature as well.

The distribution of the number of passages in the MS MARCO collection is given in Figure 8.1 by the number of terms in the passages. According to the figure, most of the passages consist of around 50 terms and nearly all of them have less than 150 terms.

A similar distribution for queries is given in Figure 8.2 that shows the distribution

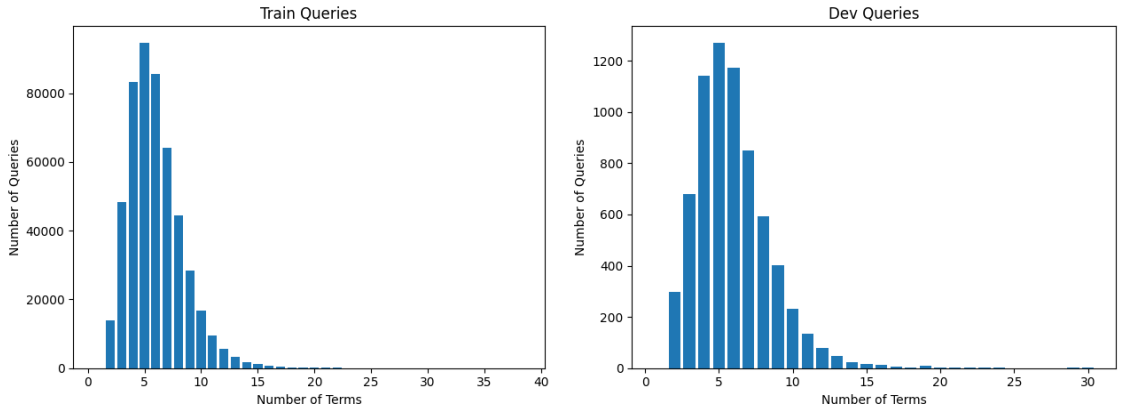

FIGURE 8.2: Histogram of the number of terms over train and dev queries.

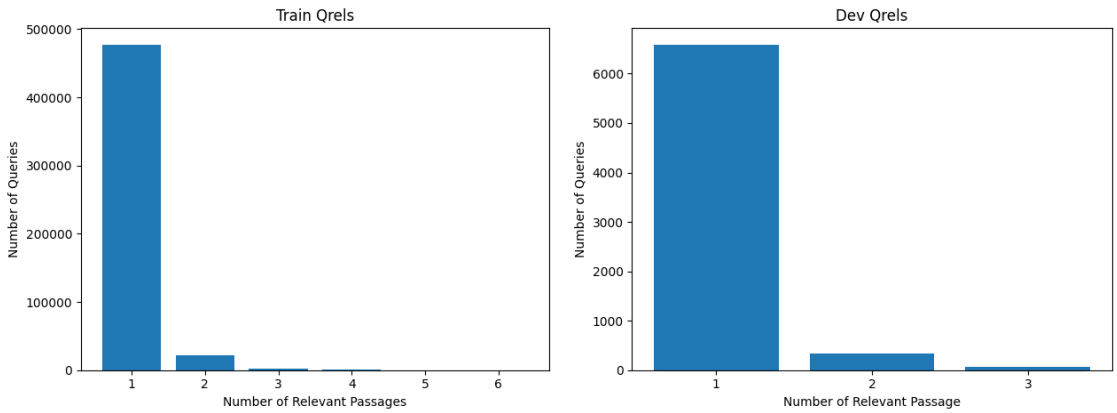

FIGURE 8.3: Histogram of the number of relevant passages over train and dev queries.

of the number of queries by the number of terms in the queries for both train and dev query sets. For both query sets, most of them consist of 3 to 10 terms.

Another distribution about queries and the number of relevant passages in the MS MARCO dataset. In Figure 8.3, the number of relevant passages are given for queries. According to this, most of the queries in both train and dev set refer to a single passage.

8.2 Configurations

BM25 has free parameters that are k_1 and b. The parameter sets for baseline, document term weighting methods, and query term weighting methods are given

TABLE 8.2: BM25 parameter configurations for methods.

Method	k_1	b
BM25	0.82	0.68
DeepCT-Index	10.0	0.90
DeepCT-Index (sqrt)	18.0	0.70
TextRank	10.0	0.90
TextRank (sqrt)	18.0	0.70
Doc2Query DocTTTTTQuery Doc2OnlyQuery DocTTTTTOnlyQuery	0.82	0.68
Oracles DeepTR DeepCT-Query BERT Models	0.82	0.68

at Table 8.2. DeepCT-Index and TextRank weight document terms by increasing occurring frequency with predicted term weight. BM25 parameters for these methods are used as specified for DeepCT[2], the parameters were fine-tuned [5]. For other methods, fine-tuned parameters[3] for baseline are used.

8.3 Methods to Evaluate

The experiment setups are required an indexed collection for query and document term weighting method to evaluate retrieval and ranking performance with the help of a text search framework.

To evaluate BM25 baseline, the original MS MARCO[4] collection indexed with Lucene[5] search engine. Query term weighting methods DeepTR, DeepCT-Query, and proposed query term weight estimation method run with the original MS MARCO collection. Query term weighting methods were trained from scratch and evaluated with MS MARCO collection for evaluations. DeepTR query term weighting was coded and trained in reference to Zheng and Callan [49]. The

[2]DeepCT, github.com/AdeDZY/DeepCT

[3]Anserini: BM25 Baseline for MS MARCO Passage Ranking, github.com/castorini/anserini/blob/master/docs/experiments-msmarco-passage.md

[4]MS MARCO - microsoft.github.io/msmarco

[5]Lucene - lucene.apache.org

Bag-of-Words approach was evaluated for single term weighting that is called DeepTR-BoW of DeepTR method and results were obtained for DeepTR-BoW. DeepCT-Query was trained with its source code[6] that provided by authors. The proposed relevance feedback and term estimation method were designed, built, and trained for the evaluations.

The collection generated with a document term weighting method has to be re-indexed to evaluate each method. The retrieval and ranking performance of these methods are evaluated by searching over the document-term-weight indexed collections. The weighted documents[7] for DeepCT-Index were downloaded and indexed with Lucene to evaluate performance. The predicted queries[8,9] for document expansion methods Doc2Query and DocTTTTTQuery, were used to expand documents, and the expanded documents were re-indexed. Document term weights were predicted by TextRank from the beginning, and documents were indexed by manipulating term frequencies to compare evaluation metrics with the other methods.

All evaluation results were produced from scratch in this thesis. Weights of the query terms are predicted with the mentioned methods in the same conditions for evaluations. Collections were indexed from scratch for TextRank with the help of the Lucene framework. For DeepCT-Index, Doc2Query, and DocTTTTTQuery, collections were indexed from estimated term weights or expansion as well as all other indexes.

[6]DeepCT, github.com/AdeDZY/DeepCT
[7]DeepCT, boston.lti.cs.cmu.edu/appendices/arXiv2019-DeepCT-Zhuyun-Dai
[8]Doc2Query, github.com/castorini/anserini/blob/master/docs/experiments-doc2query.md
[9]DocTTTTTQuery, github.com/castorini/docTTTTTquery

9. RESULTS

There are two main ideas to weighting terms for better retrieval and ranking performance by applying to queries on search time or documents on index time. In this chapter, we evaluate query term weighting and document term weighting methods in the literature. In addition, we propose a new relevance feedback method over term recall and evaluate success on the query term weighting task with other known models.

In the first section of the results, an optimal number of pairs to optimize the pairwise loss based relevance feedback method is evaluated. For the proposed relevance feedback method, we experiment with different numbers of pairs to find the winning spot for an optimal solution.

In Section 9.2, firstly we evaluate the performance of proposed new relevance feedback over the term recall. To see the applicability of the pairwise relevance feedback method in the real-life scenario, the proposed BERT-based term weight estimation model is compared to query term weighting approaches in the literature. At last, the combination of the query and document term weighting methods is investigated by using the proposed query term weighting scheme.

Document term weighting which is another main idea, explored in Section 9.3. We list document term weighting approaches that increase term frequencies or enrich documents with new terms in the literature. To create an overview, different types of approaches are evaluated on the same dataset and test cases are expanded with additional experiments.

At the end of the results, searching queries as phrases with different lengths instead of word by word is tested. We create test cases to find optimal phrase length for phrase searching. With the help of the proposed BERT-based term weight estimation model, we can predict term weights that targeted pairwise optimized relevance feedback for phrases.

Consequently, overall results are brought together into a single table to survey all evaluated approaches. Query term weighting and document term weighting methods are compared to each other and success obtained by combining these two approaches is observed.

The evaluations results for query and document term weighting methods in literature were acquired from different steps for each method. For accurate comparison and extended metrics, all results were obtained with the same index configuration, and the results were matched with reference papers.

9.1 Number of Pair for Pairwise Term Weight Optimization

In pairwise optimization, pairs are constructed from at the top of initial results to generate hard negative instances. To reduce the score of confusing irrelevant documents, hard negative samples are required. The relevant and irrelevant documents are paired by cross-product to generate pairs with hard negative instances. After ranking documents with the original queries, relevant documents are excluded from the result, and the rest of them are considered irrelevant documents. In other respect, extreme negative instances may prevent the model from being more generalized for collection. Still, there are far more completely irrelevant documents than confusing ones.

TABLE 9.1: Number of documents to optimize pairwise model.

#Docs	MRR@10	MAP@10	NDCG@5	NDCG@10	RECALL@1000
20	0.2348	0.2398	0.2513	0.2758	0.7522
100	0.2562	0.2625	0.2750	0.3066	0.8193
200	0.2621	0.2692	0.2800	0.3130	0.8406
400	0.2657	0.2723	0.2841	0.3173	0.8610
500	0.2638	0.2709	0.2825	0.3156	0.8706
600	0.2654	0.2722	0.2834	0.3167	0.8733
800	**0.2662**	**0.2733**	**0.2845**	**0.3177**	0.8784
1000	0.2633	0.2706	0.2814	0.3150	**0.8814**

To get a number of documents from the initial run that are enough to generalize the model and achieve good results, retrieval and ranking performances are compared in Table 9.1. The pairwise query term weight optimization method was evaluated with the min-max normalization. In regard to Table 9.1, the higher the number of documents, acquire higher the retrieval and ranking scores. At the point between 800 and 1000 documents, the ranking performance slightly reduced with respect to MRR@10, MAP@10, and NDCG metrics. But then, retrieval performance still increasing with respect to RECALL@1000. This spot was chosen to define the number of documents to optimize the model without missing

relevant documents with acceptable ranking performance loss. In the next experiments, 1000 documents were selected from the top of the initial results.

The number of pairs for each query can be different. Most of the queries fetch 1000 documents after initial rank. After relevant queries are excluded from the top results, the rest of the documents number differs. In the end, the number of pairs equals all relevant documents number in collection multiplied by irrelevant documents number in the results.

9.2 Query Term Weighting

The pairwise term weight optimization method is proposed as an alternative to the term recall to retrieve documents more accurately. After that, the BERT weight estimation model was trained to predict term recall and pairwise optimized term weights and compared with different methods in literature to evaluate the success of pairwise optimization and the model. First, the target term weight functions are evaluated and compared to each other, to measure the effectiveness improvement that can be attributed to the proposed target weights optimized using pairwise ranking loss. Following this, results showing the BERT based predicted term weights will be evaluated and compared to each other. Finally, we investigate the use of the proposed query term weighting method with an existing index based term weighting method.

9.2.1 Term Recall and Pairwise Term Weight Optimization

First, BM25 and oracle target query term weights are compared to each other. The baseline is the scores of the original evaluation queries. The original queries are ranked with BM25 rank algorithm and the same configuration as other methods. As can be seen by Table 9.2 and Table 9.3, the target weighting schemes can improve the baseline BM25 by more than 30% in MRR, MAP and NDCG metrics with improvements up to 15% for RECALL metrics.

The query term weighting is not applied to the queries, stated in other words, all weights are considered the same in BM25 results. The oracle scores are shown as the best possible outcome when term recall values or pairwise optimized weights are perfectly predicted. The oracle methods use the relevant and irrelevant document labels, which are not typically available at the search time. In this regard,

they can be considered as the upper bounds for the BERT based weight prediction methods. In evaluations of the oracle, term weighting schemes are applied to the evaluated queries directly. Term recall and pairwise optimized weights are extracted for the evaluation set as supervised.

Comparison of oracles which are term recall and pairwise are given in Table 9.2 and Table 9.3. According to oracle results, the pairwise term weight optimization methods achieves significantly (shown in Table 9.2 and Table 9.3 with $*$) higher scores on ranking relevant passages at the very top results as per MRR@10, MAP@10 and NDCG metrics than the term recall method.

TABLE 9.2: Query term weighting with oracle methods using the document relevance information. MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
TermRecall	0.2582	0.2661	0.7743	**0.9234**
PairWise (Min-Max)	0.2676*	0.2745*	0.7427	0.8805
PairWise (Min-Abs-Neg)	0.3086*	0.3162*	0.7867*	0.9027
PairWise (Non-Neg)	**0.3089***	**0.3164***	**0.7877***	0.8999

TABLE 9.3: Query term weighting with oracle methods using the document relevance information. NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
TermRecall	0.2781	0.3145	0.3393
PairWise (Min-Max)	0.2850*	0.3190	0.3413
PairWise (Min-Abs-Neg)	**0.3292***	0.3625*	**0.3848***
PairWise (Non-Neg)	0.3290*	**0.3628***	**0.3848***

Pairwise optimization aims to increase BM25 score of relevant documents against irrelevant ones. In this way, the ranking performance of pairwise optimization is way better than term recall. On the other hand, term recall is a statistical metric that gives which term is more important to retrieve relevant documents without

knowing irrelevant ones. As a result, term weighting by term recall can find much more relevant documents than pairwise optimization when considering wide range of ranks such as top 1000 documents, but metrics focusing on top ranks are improved with pairwise methods as it is able to discriminate relevancy in more detail as it considers irrelevant documents as well.

The weights of the query terms must be positive numbers due to constraints of the Lucene framework. To ensure that, three methods are proposed and evaluated for pairwise optimization. The first one is the fitting weights into the $[0-1]$ range. This causes at least one weight to equal to 0 and at least one weight equals to 1. The other two methods are applied to the model when optimizing weights. One of them is increasing cost when there are negative weights and the optimization model tries to minimize absolute values of the negative weights. This method still may predict negative weights and the negative weights must be assigned to the 0. The 0 assignment to the negative weights was used as a layer constraint in the third method. At the end of each iteration, negative weights are ignored and assigned to 0. This kind of projection allows us to make sure the weights are always greater or equal to 0.

According to Table 9.2 and Table 9.3, both *minimize absolute negative* and *non-negative constraints* methods achieve significantly higher performance than the *min-max normalization* method. The min-max normalization can cause result in the loss of information due to fitting the optimized weights into a fixed range. Therefore, the min-max normalized pairwise optimization had less success on the ranking and retrieval performance. The other two methods achieved very close results on performance metrics due to the 0 assignment operation to be sure non-negative weights at the end of the optimization. Consequently, the following pairwise optimization evaluations will continue with *min-max normalization* and *non-neg constraint*.

9.2.2 Term Weight Estimation

Oracle results show the possibly best results when predicted query term weights perfectly as term recall or pairwise optimization. In real life, there is no way to find exact values of term recall or pairwise term weights due to unique and unknown input queries. A way to overcome these unknown query inputs is by

creating a generalized model that learns term weights of queries by supervised and predicts term weights of new queries in the search-time.

To this end, models try to estimate term weight as close as possible to the ground truth to achieve scores as good as oracles. *DeepTR-BoW* and *DeepCT-Query* are the query term weighting approaches that use term recall as term weight, in the literature for the document ranking task. The proposed BERT based term-query interaction model is evaluated with term recall and pairwise as the target of the query term weighting task.

When the target value changes to pairwise optimized ones, the query-term interaction model acquires significantly (shown in Table 9.4 and Table 9.5 in bold*) higher scores for ranking and retrieving relevant documents for the input query than *DeepCT-Query* model. The results indicate that, target term weights learned by pairwise ranking loss independently for each query, can also be predicted by a BERT based regression model. The proposed relevance feedback achieves the best results compared to both *DeepCT-Query* and *BERT-TermRecall* in all metrics. It is interesting to note that, although *Oracle TermRecall* achieves high recall values compared to Pairwise oracle weights (Table 9.2, Table 9.3), when considering the BERT predicted weights recall of *BERT-Pairwise* is significantly better than *BERT-TermRecall*.

TABLE 9.4: Query term weighting using estimated term weights. MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
DeepTR-BoW	0.1901	0.1989	0.6845	0.8738
DeepCT-Query	0.1915	0.2005	0.6907	0.8821
BERT-TermRecall	0.1930	0.2020	0.6954	0.8837
BERT-PairWise (Min-Max)	**0.2012***	**0.2097***	0.7062*	**0.8870***
BERT-PairWise (Non-Neg)	0.2005*	0.2092*	**0.7065***	0.8842

As Table 9.4 and Table 9.5 show, *DeepCT-Query* achieves slightly higher scores than *DeepTR-BoW* on any metric, both methods try to predict term recall values of the query terms. *DeepCT-Query* tries to understand the semantic of each term in the query in considering contextual relation, but *DeepTR-BoW* simply uses vectors

TABLE 9.5: Query term weighting using estimated term weights. NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
DeepTR-BoW	0.2053	0.2380	0.2620
DeepCT-Query	0.2060	0.2393	0.2631
BERT-TermRecall	0.2080	0.2414	0.2657
BERT-PairWise (Min-Max)	**0.2179***	**0.2517***	**0.2750***
BERT-PairWise (Non-Neg)	0.2167*	0.2509*	0.2746*

of terms that measure how far from the query with direction. Due to the use of semantic information, *DeepCT-Query* is slightly better than *DeepTR-BoW* for retrieval and ranking tasks.

Pairwise optimization with non-negative constraint model got higher scores on any evaluation metrics when supervised evaluation (Oracle results, Table 9.2, Table 9.3). However, the BERT-based contextual model that tries to estimate pairwise optimized term weight, did not succeed in this increase. BERT-based model made very close estimation for both min-max normalization and non-negative constraint methods of the pairwise term weight optimization model when compare the evaluation metrics on ranking and retrieval performance.

As a final note, *DeepCT-Query*, *DeepTR-BoW* and *BERT-TermRecall* methods all try to predict Term Recall weights using different supervised regression methods. The proposed *BERT-TermRecall* method achieves the best results compared to these three methods. The proposed BERT based method and *DeepCT-Query* are very similar and achieve similar results to each other.

9.2.3 Combining with Index Term Weighting

DeepCT-Index [5] is a document term weighting method which is detailed in Section 9.3. The term frequencies are modified before indexing using the estimated term weights. The query is executed on the modified index. The proposed query term weighting method can also be applied to a re-weighted index such as *DeepCT-Index*. This strategy can combine the effects of both methods and result in superior outcomes.

Query term weights are estimated by the proposed BERT model using the modified TF and IDF values obtained from *DeepCT-Index*. The result of BM25, *DeepCT-Index*, and weighted query one is given in Table 9.6 and Table 9.7. As can be seen from the results, the proposed model improves *DeepCT-Index* significantly, with improvements in all metrics. This demonstrates that combining the proposed query term weighting method with existing index term weights can improve the results even further. The BERT-Pairwise model adds a minimal overhead to the search engine, taking less than 23 ms when processed with an RTX 2060 GPU.

TABLE 9.6: Combined effect of query term weighting with index term weighting (DeepCT-Index). MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
DeepCT-Index	0.2440	0.2516	0.7550	0.9086
DeepCT-Index + BERT-Pairwise (Min-Max)	**0.2534***	**0.2608***	**0.7732***	**0.9137***
DeepCT-Index + BERT-Pairwise (Non-Neg)	0.2524*	0.2599*	0.7721*	0.9127*

TABLE 9.7: Combined effect of query term weighting with index term weighting (DeepCT-Index). NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
DeepCT-Index	0.2624	0.2979	0.3227
DeepCT-Index + BERT-Pairwise (Min-Max)	**0.2728***	**0.3091***	**0.3335***
DeepCT-Index + BERT-Pairwise (Non-Neg)	0.2727*	0.3077*	0.3319*

Both pairwise optimization methods got close results similar to the original corpus index. The min-max normalized pairwise optimized weights are slightly better than the non-negative constraint optimized one.

9.3 Document Term Weighting

Document term weighting is done by two different approaches. One of them increasing the term frequency by repeating terms as estimated term weight, and the other approach is the expansion document with possible query terms. These approaches are evaluated under two captions and the methods that use these technics in the literature are compared to each other. Additionally, the performances of the document term expansion models are measured by only using the expansion part without the original document.

9.3.1 Term Frequency Models

In Table 9.8 and Table 9.9, BM25 results evaluation scores show the ranking and retrieval performance of running the original queries on the original corpus. When there is no manipulation on the documents and if not boost query terms, BM25 scores obtained. *DeepCT-Index* and *TextRank* change the documents and repeat the terms by their term weight estimation method. There are two different variations of both methods. The square root variant increases the resolution of the estimation by square rooting estimated term weight due to weights are between 0 and 1.

TABLE 9.8: Document term weighting. MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
DeepCT-Index	0.2440	**0.2516**	0.7550	0.9086
DeepCT-Index-sqrt	**0.2445**	**0.2516**	**0.7573**	**0.9095**
TextRank	0.1358	0.1431	0.5539	0.7738
TextRank-sqrt	0.1456	0.1535	0.5774	0.7872

According to the results, *DeepCT-Index* significantly increases the ranking and retrieval performance with respect to BM25. *DeepCT-Index-sqrt* achieved a slightly better result than *DeepCT-Index*. The square root of the term weight provides a representation of the terms in a larger space. Whenever *TextRank* reduced the performance even BM25 scores, the effect of the square root is observed.

TABLE 9.9: Document term weighting. NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
DeepCT-Index	0.2624	0.2979	0.3227
DeepCT-Index-sqrt	**0.2627**	**0.2999**	**0.3238**
TextRank	0.1459	0.1718	0.1909
TextRank-sqrt	0.1559	0.1836	0.2037

TextRank is a method to find keywords in the text. The keywords are not enough to represent the documents themselves. Therefore, *TextRank* performance is the worst one. In other respect, *DeepCT-Index* uses the contextual knowledge of the documents to weighting terms. Through, *DeepCT-Index* is more successful to estimate term weights.

9.3.2 Term Expansion Models

Term expansion-based document term weighting models are expanded original documents with possible query terms. With help of term expansion, term mismatching reduces, and adding an existing term in the document increases term frequency and it affects the document in the same way with the document term weighting models.

Doc2Query and *DocTTTTTQuery* use the same document expansion methods, but the model in *DocTTTTTQuery* is much more complex and it is a generalized generative language model. Besides, *Doc2Query* expanses documents with 10 estimated queries and *DocTTTTTQuery* expanses documents with 40 estimated queries. As a consequence, *DocTTTTTQuery* is more successful on the ranking and retrieval performance on any metrics, as shown in Table 9.10 and Table 9.11.

In Table 9.10 and Table 9.11, ranking and retrieval performances of *Doc2Query* and *DocTTTTTQuery* models are evaluated with only document expansion part. Because of the more estimated queries and better generative language model, *DocTTTTTOnlyQuery* is overwhelmingly better than *Doc2OnlyQuery* when only using expansions. Without original documents, *Doc2OnlyQuery* is worse than BM25 and *DocTTTTTOnlyQuery* reduces its ranking and retrieval performance.

TABLE 9.10: Document term expansion. MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
Doc2Query	0.2216	0.2296	0.7180	0.8928
DocTTTTTQuery	**0.2762**	**0.2845**	**0.8194**	**0.9475**
Doc2OnlyQuery	0.1290	0.1333	0.4223	0.6144
DocTTTTTOnlyQuery	**0.2634**	**0.2712**	**0.7927**	**0.9275**

TABLE 9.11: Document term expansion. NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
Doc2Query	0.2394	0.2724	0.2966
DocTTTTTQuery	**0.2996**	**0.3371**	**0.3645**
Doc2OnlyQuery	0.1372	0.1548	0.1686
DocTTTTTOnlyQuery	**0.2839**	**0.3218**	**0.3485**

9.4 Phrase Search

Lucene framework gives chance to searching phrases which are sequences of words, as a single term. The maximum length of phrases and relevance feedback estimation are retrieval and ranking performance evaluated in this section.

To clusters words into a meaningful phrase, phrases that are start or end with stopwords, are ignored. By this means, the number of phrases is reduced and possible phrases are kept in the query. Together with phrases, the original query words are used in the search. Phrases give additional information to the original query. The phrases are combined as the $n - gram$ order.

9.4.1 Maximum Length of Phrases

The first thing in the phrase search evaluation is the effect of the maximum length of the phrases. The length of the phrases is limited by 3 and the length of the input query (the whole query can be a phrase). Each phrase is added to the query if its

length is less than or equal to the limit. When the limit is defined as 3, there are phrases with lengths 2 and 3.

TABLE 9.12: Maximum length of the phrases.

Method	MRR@10	MAP@10	RECALL@1000	NDCG @10	NDCG @20
Tri-Phrase	0.1639	0.1723	0.8405	0.2041	0.2257
Max-Phrase	0.1611	0.1697	0.8382	0.2012	0.2223

In Table 9.12, *Tri-Phrase* refer to maximum length of the phrases are limited by 3, and *Max-Phrase* results addresses to the unlimited phrase length. According to the retrieval and ranking metrics, limiting the length of the phrases produces more meaningful phrase terms and gets slightly higher scores than the search results of the phrases without specifying any length limit. Hence, next phrase search evaluations are done with the limited length of the phrases, called *tri-phrase*.

9.4.2 Phrase Weighting

The query term weighting can be applied to the phrases as the same as term weighting. Weights are estimated for each phrase besides the word weights. Estimated weights are used to boost the relevance score for the word or phrase and the cumulative relevance score gives the document relevance score. BM25 score that is original queries, phrase added queries without weighting and oracle weighted phrase queries evaluated in Table 9.13 and Table 9.14.The proposed BERT-based term weight estimation model allows phrase weighting. Thanks to phrase weighting, estimated phrase weights are evaluated in the results.

In regard to results in Table 9.13 and Table 9.14, phrase search reduces retrieval and ranking performance of the original queries. When stick with the phrase search, pairwise optimization with non-negative constraint got higher retrieval and ranking performance than the term recall relevance feedback for oracle results. Pairwise optimization and min-max normalization achieved fewer scores than term recall, for this reason, when estimating term weights min-max normalized one is not included. BERT estimated pairwise optimized weights achieved approximately the same improvement on the results with word-based query term weighting (Table 9.4, Table 9.5) although still under BM25.

BERT estimations for term recall target weights are better than the estimated pairwise optimized weights according to results in Table 9.13 and Table 9.14. The results are still worst than the word-by-word search for the phrase search. The pairwise optimization for word-search achieved the highest evaluation scores and beat the term recall relevance feedback.

TABLE 9.13: Term Weighting for Phrase Query. MRR, MAP, and RECALL metrics evaluation.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000
BM25	0.1875	0.1958	0.6700	0.8575
Tri-Phrase	0.1639	0.1723	0.6212	0.8405
Oracle Tri-Phrase -TermRecall	0.2681	0.2756	0.7762	**0.9269**
Oracle Tri-Phrase -Pairwise (Min-Max)	0.2564	0.2635	0.7542	0.9050
Oracle Tri-Phrase -Pairwise (Non-Neg)	**0.2887**	**0.2971**	**0.7844**	0.8997
BERT Tri-Phrase -TermRecall	**0.1936**	**0.2025**	**0.6952**	**0.8846**
BERT Tri-Phrase -Pairwise (Non-Neg)	0.1885	0.1975	0.6906	0.8836

TABLE 9.14: Term Weighting for Phrase Query. NDCG metric evaluation.

Method	NDCG @5	NDCG @10	NDCG @20
BM25	0.2024	0.2342	0.2578
Tri-Phrase	0.1760	0.2041	0.2257
Oracle Tri-Phrase-TermRecall	0.2880	0.3219	0.3468
Oracle Tri-Phrase-Pairwise (Min-Max)	0.2753	0.3103	0.3339
Oracle Tri-Phrase-Pairwise (Non-Neg)	**0.3099**	**0.3425**	**0.3678**
BERT Tri-Phrase-TermRecall	**0.2085**	**0.2421**	**0.2657**
BERT Tri-Phrase-Pairwise (Non-Neg)	0.2027	0.2356	0.2606

9.5 Overall Results

In this section, overall results are shown in Table 9.15. The table contains baselines such as original query and document scores, query term weighting methods, document term weighting methods, document expansion methods, and finally the combination of the query and document term weighting approaches.

According to Table 9.15, document expansion models achieve the highest score on any ranking and retrieval metrics. Enriching documents reduce term mismatching between query and documents and it allow weighting for document terms in another perspective. To obtain maximum performance increment, the original documents must be kept in the index, considering the results that just using expanded part.

Applying term weighting on the document in index time gets better retrieval and ranking performance than the query term weighting. However, we combine these two approaches and achieve higher performance than the single one. We chose *DeepCT-Index* as the index and the proposed pairwise relevance feedback model as query term weighting to demonstrate the success of the combination.

As mentioned before, the proposed relevance feedback method and term weight estimation model accomplished better results than query term weight approaches in the literature. The success of the proposed pairwise relevance feedback method and combinability with the existing indices are presented in Table 9.15.

TABLE 9.15: Overall results of query and document term weighting.

Method	MRR@10	MAP@10	RECALL @100	RECALL @1000	NDCG @5	NDCG @10	NDCG @20
BM25	0.1875	0.1958	0.6700	0.8575	0.2024	0.2342	0.2578
DeepTR-BoW	0.1901	0.1989	0.6845	0.8738	0.2053	0.2380	0.2620
DeepCT-Query	0.1915	0.2005	0.6907	0.8821	0.2060	0.2393	0.2631
BERT-TermRecall	0.1930	0.2020	0.6954	0.8837	0.2080	0.2414	0.2657
BERT-Pairwise (Min-Max)	**0.2012**	**0.2097**	0.7062	**0.8870**	**0.2179**	**0.2517**	**0.2750**
BERT-Pairwise (Non-Neg)	0.2005	0.2092	**0.7065**	0.8842	0.2167	0.2509	0.2746
DeepCT-Index	0.2440	**0.2516**	0.7550	0.9086	0.2624	0.2979	0.3227
DeepCT-Index-sqrt	**0.2445**	**0.2516**	**0.7573**	**0.9095**	**0.2627**	**0.2999**	**0.3238**
TextRank	0.1358	0.1431	0.5539	0.7738	0.1459	0.1718	0.1909
TextRank-sqrt	0.1456	0.1535	0.5774	0.7872	0.1559	0.1836	0.2037
Doc2Query	0.2216	0.2296	0.7180	0.8928	0.2394	0.2724	0.2966
DocTTTTTQuery	**0.2762**	**0.2845**	**0.8194**	**0.9475**	**0.2996**	**0.3371**	**0.3645**
Doc2OnlyQuery	0.1290	0.1333	0.4223	0.6144	0.1372	0.1548	0.1686
DocTTTTTOnlyQuery	0.2634	0.2712	0.7927	0.9275	0.2839	0.3218	0.3485
DeepCT-Index + Pairwise (Min-Max)	**0.2534**	**0.2608**	**0.7732**	**0.9137**	**0.2728**	**0.3091**	**0.3335**
DeepCT-Index + Pairwise (Non-Neg)	0.2524	0.2599	0.7721	0.9127	0.2727	0.3077	0.3319

10. CONCLUSION

A text search engine works with a huge amount of corpus, and it keeps text data in a structured database to retrieve as fast as possible when queried by a user request. Search engines use retrieval and ranking algorithms when searching a query to find the best candidates that meet the user request in the vast dataset. To take advantage of a search engine in all possible ways, neural network solutions are applied to documents and queries. While doing this, minimal overhead has to be added to the search cost to ensure a fluent user experience.

The usage of neural networks to improve retrieval and ranking performance focus on two sides of the query searching task. The first one is editing documents in index-time and the second one is weighting query terms in search-time. The index-time operations are offline and its cost does not affect the execution time of the search. However, weighting query terms adds an additional operation that estimates term weights to the search-time.

Semantic knowledge is the key to the predicting term weights of a document or a query. The language models that are well known were surveyed to understand principles of work, architecture, and their application for this task, in the thesis. Statistical relevance feedback method called Term Recall and pairwise weight optimization which was developed for better relevance feedback method were detailed in this study.

In this thesis, we evaluated the application of neural networks in index-time and search-time as a ranking method by weighting terms. We created a benchmark for document term weighting and document expanding methods in the literature. In addition, we compared query term weighting approaches with our model and we proposed a relevance feedback method that uses pairwise ranking loss to find optimal term weights for the input query.

Considering experiments that evaluate retrieval and ranking performance with several metrics, the query and document term weighting is a decent way to retrieve more accurate documents for the input query and rank properly by a relevance metric like BM25. When we compare the best results of the usage of the neural network in search-time and index-time, the document term weighting methods in the index-time got better results than query term weighting in the

search-time. One of the advantages of these two approaches is they can be used together in the same search engine. According to the experiments, the combination of the query and document term weighting significantly exceeded to only using query or document term weighting.

The actual contribution of this work is the utilization of the well-known pairwise loss function for the term weighting as relevance feedback besides comparing term weighting approaches in the literature. The pairwise relevance feedback method finds optimal weights that boost the contribution of essential terms for relevant documents when throwing back the irrelevant ones. The proposed relevance feedback method outperformed the statistical term recall. To support our work, we applied the pairwise optimized term weights estimated by the BERT-based regression model to queries in search-time with minimal cost. The proposed relevance feedback method increased retrieval and ranking performance significantly compared to term recall approaches.

Pairwise term weight optimization requires lots of pairs for generalization and hard negative instances for an optimal solution. A query can fetch more than a thousand documents, and the initial ranking of documents consists of mostly hard negative instances. To this end, the proposed relevance feedback method was demonstrated on the query term weighting task, and the strategy was evaluated from various perspectives.

The combination of query term expansion models and the proposed query term weighting is considered as future work of this thesis. Evaluating the proposed relevance feedback method on the document term weighting is an additional task for follow-up works. Additionally, studies can increase the predictability of relevance feedback to catch oracle results.

REFERENCES

[1] Sergey Brin and Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: *Computer Networks* 30 (1998), pp. 107–117.

[2] Christopher JC Burges. "From ranknet to lambdarank to lambdamart: An overview". In: *Learning* 11.23-581 (2010), p. 81.

[3] Yinqiong Cai et al. "Semantic Models for the First-stage Retrieval: A Comprehensive Review". In: *CoRR* abs/2103.04831 (2021). arXiv: 2103.04831.

[4] Zhe Cao et al. "Learning to Rank: From Pairwise Approach to Listwise Approach". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, 129–136. ISBN: 9781595937933. DOI: 10.1145/1273496.1273513.

[5] Zhuyun Dai and Jamie Callan. "Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval". In: *CoRR* abs/1910.10687 (2019). arXiv: 1910.10687.

[6] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[7] Yoav Freund et al. "An E cient Boosting Algorithm for Combining Preferences". In: *International Conference on Machine Learning, Madison WI*. Citeseer. 1998.

[8] Luyu Gao, Zhuyun Dai, and Jamie Callan. "COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List". In: *CoRR* abs/2104.07186 (2021). arXiv: 2104.07186.

[9] Jiafeng Guo et al. "A Deep Relevance Matching Model for Ad-Hoc Retrieval". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA:

Association for Computing Machinery, 2016, 55–64. ISBN: 9781450340731. DOI: 10.1145/2983323.2983769.

[10] Shuguang Han et al. "Learning-to-Rank with BERT in TF-Ranking". In: *CoRR* abs/2004.08476 (2020). arXiv: 2004.08476.

[11] Baotian Hu et al. "Convolutional neural network architectures for matching natural language sentences". In: *Advances in neural information processing systems* 27 (2014), pp. 2042–2050.

[12] Po-Sen Huang et al. "Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data". In: *Proceedings of the 22nd ACM International Conference on Information |& Knowledge Management*. CIKM '13. San Francisco, California, USA: Association for Computing Machinery, 2013, 2333–2338. ISBN: 9781450322638. DOI: 10.1145/2505515.2505665.

[13] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Trans. Inf. Syst.* 20.4 (Oct. 2002), 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418.

[14] Thorsten Joachims. "Optimizing Search Engines Using Clickthrough Data". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, 133–142. ISBN: 158113567X. DOI: 10.1145/775047.775067.

[15] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2020, 39–48. ISBN: 9781450380164.

[16] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[17] Antonio Mallia et al. "Learning Passage Impacts for Inverted Indexes". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '21. Virtual Event, Canada: Association for Computing Machinery, 2021, 1723–1727. ISBN: 9781450380379.

[18] Rada Mihalcea and Paul Tarau. "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411.

[19] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013.

[20] I. C. Mogotsi. "Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: Introduction to information retrieval". In: *Information Retrieval* 13.2 (Sept. 2009), pp. 192–195. DOI: 10.1007/s10791-009-9115-y.

[21] Tri Nguyen et al. "MS MARCO: A Human Generated MAchine Reading COmprehension Dataset". In: *CoRR* abs/1611.09268 (2016). arXiv: 1611.09268.

[22] Rodrigo Nogueira and Kyunghyun Cho. "Passage Re-ranking with BERT". In: *CoRR* abs/1901.04085 (2019). arXiv: 1901.04085.

[23] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. "From doc2query to docTTTTTquery". In: *Online preprint* (2019).

[24] Rodrigo Nogueira et al. "Document Expansion by Query Prediction". In: *CoRR* abs/1904.08375 (2019). arXiv: 1904.08375.

[25] Rodrigo Nogueira et al. "Multi-Stage Document Ranking with BERT". In: *CoRR* abs/1910.14424 (2019). arXiv: 1910.14424.

[26] Liang Pang et al. "A study of matchpyramid models on ad-hoc retrieval". In: *arXiv preprint arXiv:1606.04648* (2016).

[27] Liang Pang et al. "DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17. Singapore, Singapore: Association for Computing Machinery, 2017, 257–266. ISBN: 9781450349185. DOI: 10.1145/3132847.3132914.

[28] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[29] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[30] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683.

[31] S. E. Robertson and K. Sparck Jones. "Relevance weighting of search terms". In: *Journal of the American Society for Information Science* 27.3 (May 1976), pp. 129–146. DOI: 10.1002/asi.4630270302.

[32] S. E. Robertson and S. Walker. "Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval". In: *SIGIR '94*. Ed. by Bruce W. Croft and C. J. van Rijsbergen. London: Springer London, 1994, pp. 232–241. ISBN: 978-1-4471-2099-5.

[33] Stephen Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019.

[34] Stephen E Robertson et al. "Okapi at TREC-3". In: *Nist Special Publication Sp* 109 (1995), p. 109.

[35] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682.

[36] Aliaksei Severyn and Alessandro Moschitti. "Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: Association for Computing Machinery, 2015, 373–382. ISBN: 9781450336215. DOI: 10.1145/2766462.2767738.

[37] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-Attention with Relative Position Representations". In: *CoRR* abs/1803.02155 (2018). arXiv: 1803.02155.

[38] Noam Shazeer and Mitchell Stern. "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4596–4604.

[39] Yelong Shen et al. "Learning Semantic Representations Using Convolutional Neural Networks for Web Search". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14 Companion. Seoul, Korea: Association for Computing Machinery, 2014, 373–374. ISBN: 9781450327459. DOI: 10.1145/2567948.2577348.

[40] K. Sparck Jones, S. Walker, and S.E. Robertson. "A probabilistic model of information retrieval: development and comparative experiments: Part 1". In: *Information Processing & Management* 36.6 (2000), pp. 779–808. ISSN: 0306-4573. DOI: https://doi.org/10.1016/S0306-4573(00)00015-7.

[41] K Sparck Jones, S Walker, and S.E Robertson. "A probabilistic model of information retrieval: development and comparative experiments: Part 2".

In: *Information Processing & Management* 36.6 (2000), pp. 809–840. ISSN: 0306-4573. DOI: https://doi.org/10.1016/S0306-4573(00)00016-9.

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[43] Andrew Trotman, Antti Puurula, and Blake Burgess. "Improvements to BM25 and Language Models Examined". In: *Proceedings of the 2014 Australasian Document Computing Symposium*. ADCS '14. Melbourne, VIC, Australia: Association for Computing Machinery, 2014, 58–65. ISBN: 9781450330008. DOI: 10.1145/2682862.2682863.

[44] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008.

[45] Shengxian Wan et al. "Match-srnn: Modeling the recursive matching structure with spatial rnn". In: *arXiv preprint arXiv:1604.04378* (2016).

[46] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[47] Jun Xu and Hang Li. "AdaRank: A Boosting Algorithm for Information Retrieval". In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '07. Amsterdam, The Netherlands: Association for Computing Machinery, 2007, 391–398. ISBN: 9781595935977. DOI: 10.1145/1277741.1277809.

[48] Hamed Zamani et al. "From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Torino, Italy: Association for Computing

Machinery, 2018, 497–506. ISBN: 9781450360142. DOI: 10.1145/3269206.3271800.

[49] Guoqing Zheng and Jamie Callan. "Learning to Reweight Terms with Distributed Representations". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: Association for Computing Machinery, 2015, 575–584. ISBN: 9781450336215. DOI: 10.1145/2766462.2767700.

[50] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. "Inverted Files versus Signature Files for Text Indexing". In: *ACM Trans. Database Syst.* 23.4 (Dec. 1998), 453–490. ISSN: 0362-5915. DOI: 10.1145/296854.277632.