KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science

Second cycle, 30 credits

Re-ranking search results with KB-BERT

BJARKI VIÐAR KRISTJÁNSSON

Stockholm, Sweden, 2022

Re-ranking search results with KB-BERT

BJARKI VIÐAR KRISTJÁNSSON

Master's Programme, Computer Science, 120 credits
Date: May 12, 2022

Supervisors: Johan Boye, Richard Hartman
Examiner: Joakim Gustafsson
School of Electrical Engineering and Computer Science
Host company: Findwise
Swedish title: Omrankning av sökresultat med KB-BERT

Abstract

This master thesis aims to determine if a Swedish BERT model can improve a BM25 search by re-ranking the top search results. We compared a standard BM25 search algorithm with a more complex algorithm composed of a BM25 search followed by re-ranking the top 10 results by a BERT model. The BERT model used is KB-BERT, a publicly available neural network model built by the National Library of Sweden. We fine-tuned this model to solve the specific task of evaluating the relevancy of search results.

A new Swedish search evaluation dataset was automatically generated from Wikipedia text to compare the algorithms. The search evaluation dataset is a standalone product and can be beneficial for evaluating other search algorithms on Swedish text in the future.

The comparison of the two algorithms resulted in a slightly better ranking for the BERT re-ranking algorithm. These results align with similar studies using an English BERT and an English search evaluation dataset.

Keywords

Natural language processing, Information retrieval, BERT, KB-BERT, Search evaluation

Sammanfattning

Denna masteruppsats syftar till att avgöra om en svensk BERT-modell kan förbättra en BM25-sökning genom att ranka om de bästa sökresultaten. Vi jämförde en standard BM25-sökalgoritm med en mer komplex algoritm som består av en BM25-sökning följt av omrankning av de 10 bästa resultaten med en BERT-modell. BERT-modellen som används är KB-BERT, en allmänt tillgänglig neural nätverksmodell byggd av Kungliga biblioteket. Vi finjusterade den här modellen för att lösa den specifika uppgiften att utvärdera sökresultatens relevans.

En ny svensk datamängd för utvärdering av sökresultat genererades automatiskt från Wikipedia-text för att jämföra algoritmerna. Datamängden är en fristående produkt och kan vara till nytta för att utvärdera andra sökalgoritmer på svensk text i framtiden.

Jämförelsen av de två algoritmerna resulterade i en något bättre rankning för BERT-omrankningsalgoritmen. Dessa resultat överensstämmer med liknande studier som använder en engelsk BERT och en engelsk datamängd för utvärdering av sökresultat.

Nyckelord

Naturlig språkbehandling, Informationssökning, BERT, KB-BERT, Sökutvärdering

Acknowledgments

I would like to thank...

Johan Boye for forming the initial project idea and for guiding me throughout the project,

Richard Hartman for helping me create a project plan, giving me resources for the project work and incorporating me into the Findwise team,

Findwise for making me feel welcome and providing me with cloud computing credits,

Joakim Gustafsson for examining my thesis,

Sigurlaug for giving me feedback on the writing and always being there for me,

Family and friends for overall support throughout my studies.

Stockholm, May 2022
Bjarki Viðar Kristjánsson

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Information retrieval

In computer science, information retrieval plays a key role. Information retrieval is the process of finding relevant information in large datasets. A user has some information need, and a system has to satisfy this need by giving the user relevant information. There are different ways to solve this problem. Typically the user has to invent a query that represents their information need. A query usually is a series of words such as: ”the capital of Sweden”. This query is then passed to the information retrieval system. When someone types something into Google, they create a query that is supposed to represent their information need. The information retrieval system then takes the user's query, maybe changes it somehow, and compares it to a representation of its data or documents. After this comparison, the system returns some information that it thinks will satisfy the user.

1.1.2 The ranking problem

When an information retrieval system returns its results, it has another problem to figure out. It may have some good documents to show to the user, but how is the system supposed to tell which documents are the most relevant? This problem is called the ranking problem. Usually, this is solved by giving each document a calculated score for how well it corresponds to the typed query. Then when the system returns the results, it will rank them in the order of how high the score is: the higher the score, the higher the similarity to the query.

We should then get the search result with the highest similarity on the top of the list.

A search engine needs to have a good scoring function. The scoring function decides what the top search results will be, and it has been shown that people rarely pay attention to the results that are not up at the top[1].

1.1.3 Different search algorithms

Search algorithms vary. The most simple ones are word-based algorithms. As an example, a word-based algorithm could return results like the one shown in figure 1.1.

Query: ”dog”
Search results:

1. A hot dog is a fast food. Common hot dog sausage ingredients include...
2. The dog is a domesticated descendant of the wolf...
3. There are many different shapes for dog tails: straight, straight up...

Figure 1.1: *A possible search result ranking of a word based search algorithm.*

We see in the example in figure 1.1 that the top result is not likely to satisfy the information need of the user. This result is about the fast food ”hot dog”. However, the user probably wanted information about the animal. The simple algorithm just sees the correct word ”dog” and thinks this is a good result.

There are types of algorithms that can deal with a problem like this. Those are algorithms that take the context into account. See in figure 1.2

The search results in figure 1.2 are more relevant than the results in figure 1.1. A context-based ranking algorithm can even return a result that does not necessarily have the same word as the query. This can be seen in the second result in figure 1.2 where only the plural of the query ”dog” appears.

If the context is taken into account, the search results will likely improve. An example of a word-based algorithm is an algorithm called BM25. There is also a way of interpreting words in context through another algorithm called BERT. The goal of this project will be to see if the context-based BERT can improve the word-based BM25.

Query: ”dog”
Search results:

1. The dog is a domesticated descendant of the wolf...
2. Dogs are the most variable mammal on earth with...
3. There are many different shapes for dog tails: straight, straight up...

Figure 1.2: *A possible search result ranking of a context based search algorithm.*

Evaluating search algorithms In this project, we will be comparing different search algorithms. A way to do that is to create a correct ranking of the search results for a particular query. This ranking can be called a ground truth ranking and will be generated by a search evaluation dataset. The way such a dataset is created will be explained in section 3.1. Any ranking from an algorithm for a particular query can be compared to the ground truth ranking for that same query, resulting in a score of how well the algorithm did. This is explained more thoroughly in section 2.2.

1.2 Research question

Will a ranking method combining BM25 similarity ranking[2] and a Swedish BERT model[3] perform a better search ranking than just the BM25 ranking alone?

1.3 Objective

The objective of the project is divided into two components.

- Create a search evaluation dataset in Swedish.
- Compare the performance of a ranking method combining BM25 similarity ranking and a Swedish BERT model with a simple BM25 ranking algorithm.

1.4 Contributions

It is essential for people researching information retrieval to have a shared dataset to evaluate their search engines. The search evaluation dataset created in this project could become useful for people that want to evaluate their search algorithms in Swedish text[4]. The dataset was a big part of the project and will hopefully become helpful for further research in the field. Also, the results from the search algorithm comparison can become valuable. We are unaware that the Swedish BERT has been used in this setting before. So it can show the value of the algorithm on a new task[5].

An important thing to note is that, in essence, this project follows a method of another paper called *Passage re-ranking with BERT*[6]. We aim to do the same experiment but only in Swedish, not English. So one of the contributions of this thesis is to investigate if the results from the original paper can carry over to Swedish.

1.5 Delimitations

The project only incorporates these two tasks:

- Creation of a Swedish search evaluation dataset
- Comparing a neural network algorithm against a standard ranking algorithm

In the second task, we will use a BERT language model from The National Library of Sweden[3]; we will not create our own. The standard ranking algorithm will also not be created. We will use the default ranking algorithm in Elasticsearch[7].

Chapter 2

Background

This chapter equips the reader with the necessary knowledge to read the rest of the paper. Here we explain the theory and the concepts in the field of this thesis.

2.1 The BM25 ranking algorithm

One of the most successful search ranking algorithms is the BM25 algorithm[2]. This algorithm uses a bag-of-words approach to the ranking. Bag-of-words means neither the order of the words in the document nor the order of the words in the query matter. So if we search for ”Elon Musk” or ”Musk Elon” we will get the same results with the same ranking order. Let us look at the formula for the algorithm. It gives a score to a document d in the following way:

$$bm25(d) = \sum_{t \in q} \log\left(\frac{N}{df_t}\right) \frac{(k+1)tf_{td}}{k(1-b+b\frac{L_d}{L_{ave}}) + tf_{td}}$$

Here, $bm25(d)$ is the score the algorithm will give document d. When we search with the query ”Stockholm restaurants” the algorithm calculates this $bm25(d)$ score for every single document in the text corpus we are searching in. We represent the query with the letter q and every term in the query with t. For the mentioned query, we have two terms: ”Stockholm” and ”restaurants”. When calculating the score, we sum all the term scores together. The term $\log\left(\frac{N}{df_t}\right)$ is called inverse document frequency and conveys the informativeness of searching with this query term. N is the number of documents in our corpus, and df_t is how many documents contain the term t. For instance, we do not gain much information by searching for ”the” because almost all documents

have this word. So the informativeness of that search is low. In this case, $N \approx df_t$ and therefore the result of the logarithm will approach zero, meaning not so informative search. The term L_d and L_{ave} correspond to the length of document d and the average document length, respectively. If document d is long, the score will be negatively affected. Then we have tf_{td}, which is the term frequency of term t in the document. If tf_{td} is high, the score will be positively affected. At last, we have the b and k parameters. The b controls how much of a role the document length plays in the final score. If we set $b = 0$, the document length will not matter for the final score. On the other hand, k controls how quickly an increase in term frequency results in term-frequency saturation. That is, if the word ”Stockholm” appears five times in a document, maybe it will not matter that much if it appears five more times. The document is still equally relevant. In a way, k controls how often a term has to appear until more term appearances will no longer affect the score.

2.2 Evaluating ranking

Now we have seen a ranking algorithm, but how do we know if it is any good? One of the reasons the ranking problem is so complex is that the optimal solution is not always clear. It is hard to know what to maximize for.

A typical framing of this problem is to approach it from a user happiness standpoint. Then we need a way of quantifying user happiness. We want the user to finish their task and not waste unnecessary time. So we have to give them the most relevant search results. Relevance is often used as a substitute for user happiness in the field. However, relevance is also hard to measure. To solve this, people have created all sorts of benchmark datasets. Judges will manually go through the documents, and for every query, they will give each document a score. The score is supposed to tell how relevant the document is. It is similar to when we had the BM25 algorithm tell us how relevant a document was. However, now we are creating a ground truth relevance score because the judges are humans and should measure their satisfaction with each search result.

There are different ways in which the judges might evaluate the search results. One way is to label each document with 0 or 1 depending on if it is irrelevant or relevant. Another method is to label with a non-binary judgment. It means we can choose a score for each document from more than two classes. For example, we can have the scores from zero to three[8]:

- (0): The document does not contain any information about the topic.

- (1): The document only points to the topic. It does not contain more or other information than the topic description.
- (2): The document contains more information than the topic description but the presentation is not exhaustive.
- (3): The document discusses the themes of the topic exhaustively.

This evaluation method will result in a more robust ranking as we have a more accurate classification of documents than the binary method has. Now, when we have a method that evaluates every document, we can look at different measures to evaluate the final ranked results.

2.2.1 Cumulative Gain

When an algorithm has ranked all the documents for a particular search query, we want to know how it performed. One measure is Cumulative Gain. Let us say that all documents have been labeled with labels: 0, 1, 2, or 3 for the current query. We can then look at the Cumulative Gain at rank p. In essence, we sum up the relevance scores of the first p documents:

$$CG_p = \sum_{i=1}^{p} rel_i$$

Here, rel_i is the relevance score 0, 1, 2, or 3 for document at rank i. That is, when $i = 1$, we are looking at the top result. The higher the Cumulative Gain CG_p, the better the ranking.

2.2.2 Discounted Cumulative Gain

Another method which improves the Cumulative Gain method is Discounted Cumulative Gain or DCG_p. This method penalises the ranking if relevant documents are delivered late:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2 (i+1)}$$

The $\log_2 (i+1)$ term works as a penalization for the high relevance documents if they are ranked low; that is, the value of i is high. Now, if we are looking at Discounted Cumulative Gain at rank $p = 10$, the order of our documents will matter, unlike in the Cumulative Gain. We will get a higher

score if the algorithm serves a document with a relevance score of 3 as our top result than if it places it in position 10.

2.2.3 Normalized Discounted Cumulative Gain

An even more improved method is the Normalized Discounted Cumulative Gain method. It is mainly for the purpose of comparing how our algorithm performs on different queries. The Normalized Discounted Cumulative Gain will normalize the score of the algorithm's ranking with the perfect ranking score. It is done by calculating first the Ideal Discounted Cumulative Gain, which is the DCG_p for the ideal ranking of the documents. The ideal ranking will look like this:

rank	document	relevance score
1	doc_c	3
2	doc_a	3
3	doc_f	2
4	doc_b	1
5	doc_d	1
6	doc_e	0

That is, the relevance score are descending as we go through the documents. On the other hand, our algorithm might make a ranking such as:

rank	document	relevance score
1	doc_c	3
2	doc_f	2
3	doc_a	3
4	doc_e	0
5	doc_d	1
6	doc_b	1

The Ideal Discounted Cumulative Gain or $IDCG_p$ will always be more or equal to DCG_p. Now, the Normalized Discounted Cumulative Gain at rank p is:

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

So if we have perfect ranking, $NDCG_p = 1$. As mentioned above, the $NDCG_p$ gives us the ability to compare the ranking of different queries. If the $NDCG_p$ is higher for the query "Stockholm restaurants" than for "giant

slalom," we can infer that our algorithm is better at finding relevant documents about eateries in Stockholm than serving up relevant documents about alpine skiing.

2.2.4 TREC CAR

The Text Retrieval Conference (TREC) is held every year by the National Institute of Standards and Technology in the U.S. and the U.S. Department of Defense. They started these conferences 30 years ago to support research in information retrieval. In 2017, part of the conference was about Complex Answer Retrieval (CAR). The same year, the TREC CAR dataset was created. The dataset includes texts from the English Wikipedia, including titles, headers, and paragraphs. How the dataset is constructed is explained in detail in chapter 3, where we implement a similar dataset in Swedish.

In 2017 when this dataset was introduced, one of the goals for it was to be used to evaluate other algorithms in a so-called "passage task"[9]. In the TREC CAR paper, the passage task was defined as follows: *Given an outline for complex topic outline Q, retrieve for each of its sections H_i, a ranking of relevant passages S.* In essence, we input a search query containing the Wikipedia title and heading into a search engine. The engine has to output a paragraph with this heading only by knowing the text of the paragraph but not the headings.

This paper attempts to create a similar dataset but using Swedish Wikipedia articles.

2.3 Natural language processing

Natural language processing or NLP is a subfield of computer science. By natural language, we mean text or speech that humans understand. NLP is about how a computer processes such data. The goal is to make computers perform interesting tasks involving natural language. Use cases for this kind of technology include machine translation between languages, chatbots, and automatic text summarization. NLP is a fast-moving field, and significant progress has been made in recent years. Research papers introducing new methods or improvements to older methods pile up every week.

2.3.1 Word embeddings

It is not trivial to program a computer to perform advanced natural-language tasks. We humans have some mechanism that models words to ideas. When we read a word, an image or an idea often appears in our heads. For the most part, we are not aware of how this is done, but our carbon-based computer called the brain takes care of this. For a computer to do something similar, researchers have come up with a method called *word embeddings*[10].

The idea is that we represent every word as a vector in a high-dimensional space. So for example the word ”dog” could get the vector:

$$v_{dog} = \begin{bmatrix} 1.02 \\ -3.80 \\ \vdots \\ 0.46 \end{bmatrix}$$

Each word will have its representation in vector space. One way to create vectors like this is to let the word's context decide the vector values. In a vast corpus of texts, each word comes up multiple times. Some words will usually appear in similar contexts and, as a result, will get similar vectors. For example, it is common to say ”I drive a car” and ”I drive a truck”. Therefore, it is likely that the words ”car” and ”truck” will get similar vectors in vector space.

Dot product of word embeddings It is possible to compare word similarity by calculating the dot product of two words. If two vectors are of the same length and the dot product between them is large, it means there is a high similarity between them. More often than not, the word embeddings are of similar length, so the dot product between two embeddings can certainly contain information about the similarity of the two words.

2.3.2 Self-Attention

A relatively new approach for processing language is something called an Attention mechanism[11]. This mechanism revolves around making the machine understand how words relate to each other and understanding context in natural language. To comprehend it better, let us look at one type of Attention mechanism called Self-Attention.

When confronted with the word ”right”, there are multiple ways one can interpret it. Are we talking about the noun corresponding to a direction as in

”left or right”? Or are we talking about the noun as in ”human right”? There are even more possibilities like the adjective ”right” in ”The right thing to do is...”. How do we know which meaning to give to the word? The answer lies in the context of the word.

In order to understand the meaning conveyed by a word, we have to look at the words accompanied by it. The Self-attention mechanism essentially does this by processing the word embeddings of the words in the text and figuring out connections between them. The mechanism uses multiple calculations that will be explained here step-by-step.

Using the dot product of word embeddings First, we must understand that the word embeddings are the heart of everything done by the Self-attention mechanism. These vectors encode the meaning of each word, and as discussed in section 2.3.1, word similarity can be calculated using these embeddings.

However, we do not only want to find out how similar words are. It is also interesting to discover other kinds of connections between words. For example, the connection between a subject and a verb can be interesting. Alternatively, the relation between a preposition and its complement can give us some insight. To make connections like these in a sentence, we have to take different measures.

Scaled dot-product self-attention The Scaled dot-product self-attention is a method to discover new connections between words beyond their similarity. In this method, every word embedding goes through three different linear projections that create three different word representations. We call these new representations key, value, and query vectors. The linear projection is simply a matrix multiplied by the word embedding. All in all, we have three matrices, one for key projections, one for value projections, and one for query projections. For example, if we have the token ”book” with the embedding v_{book}, we will multiply v_{book} with the three matrices to acquire the key, value, and query for ”book”:

$$
\begin{aligned}
key_{book} &= v_{book} M_{key} \\
value_{book} &= v_{book} M_{value} \\
query_{book} &= v_{book} M_{query}
\end{aligned}
$$

where M_{key}, M_{value} and M_{query} are the linear projection matrices. The v_{book} is a word embedding for ”book” and the key_{book}, $value_{book}$, and $query_{book}$

are the key, value, and query vectors for ”book”. These three different projections of the word embedding allow the algorithm to focus on different attributes of the vector. Note that the projection matrices are initialized randomly, so it is not like the algorithm knows what attributes it is searching for.

We can now do some calculations involving queries, keys and values. Let us look at the example sentence ”He likes books”. We generate the key, value, and query for every word. Next, we focus on the first word ”He”, and take these steps:

1. Take dot products of $query_{He}$ and the key of every word in the sentence. There will be three dot products:

$$query_{He} \cdot key_{He}$$
$$query_{He} \cdot key_{likes}$$
$$query_{He} \cdot key_{books}$$

2. Scale the result of the dot products by the dimension of the key:

$$\frac{query_{He} \cdot key_{He}}{\sqrt{d_k}}$$
$$\frac{query_{He} \cdot key_{likes}}{\sqrt{d_k}}$$
$$\frac{query_{He} \cdot key_{books}}{\sqrt{d_k}}$$

3. Put the results of the scaling through a softmax function. The softmax function takes in a vector with the three values above and outputs a probability distribution.

$$softmax(\begin{bmatrix} \frac{query_{He} \cdot key_{He}}{\sqrt{d_k}}) \\ \frac{query_{He} \cdot key_{likes}}{\sqrt{d_k}}) \\ \frac{query_{He} \cdot key_{books}}{\sqrt{d_k}}) \end{bmatrix}) = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}, \text{ where } p_1 + p_2 + p_3 = 1$$

So we get three probabilities that sum up to one. These are the probabilities for how much the word ”He” attends to the words ”He”,

”likes”, and ”books”. In other words, how much one word attends to another should convey how strong a particular relationship is between the two words in question. However, the relationship is not defined in any way. It could be any type of relationship; this depends on the initialization of the linear projection matrices that created the keys and the queries in the beginning. It also depends on the subsequent training. It is good to note that the softmax function makes large numbers appear more significant, and low numbers will become minuscule.

4. Finally, we take the three softmax results we have and multiply them with the corresponding value vector. So we will have:

$$p_1 * value_{He}$$
$$p_2 * value_{likes}$$
$$p_3 * value_{books}$$

This will result in three vectors. We add all three vectors from the sentence together into one final vector. This vector can be called a contextualized embedding of the word ”He” in the sentence ”He likes books”. It should describe in some way how the word ”He” relates to itself and the other two words in the sentence.

This four-step process is repeated for the other words ”likes” and ”books”. In practice, this can be done more effectively by packing the vectors into a matrix. The queries, keys, and values will be packed into matrices Q, K, and V[12]. Then these four steps are all performed in a single calculation:

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

This calculation can be visualized as in figure 2.1.

2.3.3 Transformer

The Transformer was introduced in 2017 in a seminal paper called ”Attention is all you need”[12]. In the paper, researchers at Google introduced the Transformer, which is solely based on the self-attention mechanism to process language.

The Scaled dot-product self-attention discussed in the previous chapter is the building block of the Transformer. However, it is used with some added

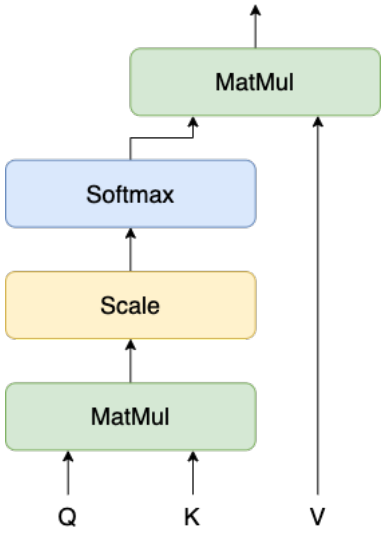

Figure 2.1: *Scaled dot-product self-attention visualized.*

features. In the Transformer, it is called Multi-Head Attention. It does the Scaled dot-product self-attention multiple times with different projections for the keys, values, and queries. For a thorough explanation, the original paper covers it well[12].

There are so many different types of relationships words can have that we want the Transformer to learn. Doing multiple attention calculations with different projections for the key, value, and queries gives the machine a better opportunity to learn varying connections between words.

The Transformer uses so-called encoder-decoder architecture. It means that the input goes into an encoder that uses Self-Attention, as discussed in section 2.3.2 to encode the input. Then the decoder takes that encoding and generates an output in the desired format. For example, if the Transformer implements a translation model, the input could be a sentence in English, and the output becomes the same sentence in Swedish. Let us dive deeper into the architecture.

Positional encoding Assume the Transformer is a language translation model. We start with a sentence in English. The first thing the Transformer does is to take the sentence's word embeddings and change them using positional encoding. The positional encoding adds extra information to the word embeddings, information about where the words lie in the input sentence. With these details, the relationships between the words in the sentence become more apparent to the Transformer.

Encoder Next the sentence enters the encoder, see figure 2.2. The encoder is built of N identical layers. In each layer, there is a Multi-Head Attention block and a fully connected feed-forward neural network block. For both blocks, there is a residual connection[13] employed around the block, and a layer normalization is executed afterward. Residual connections make it easier to train deep neural networks[13]. After going through all N layers of the encoder, the vectors will have changed drastically. The new vectors will have information encoded in them, which was extracted with the attention mechanism.

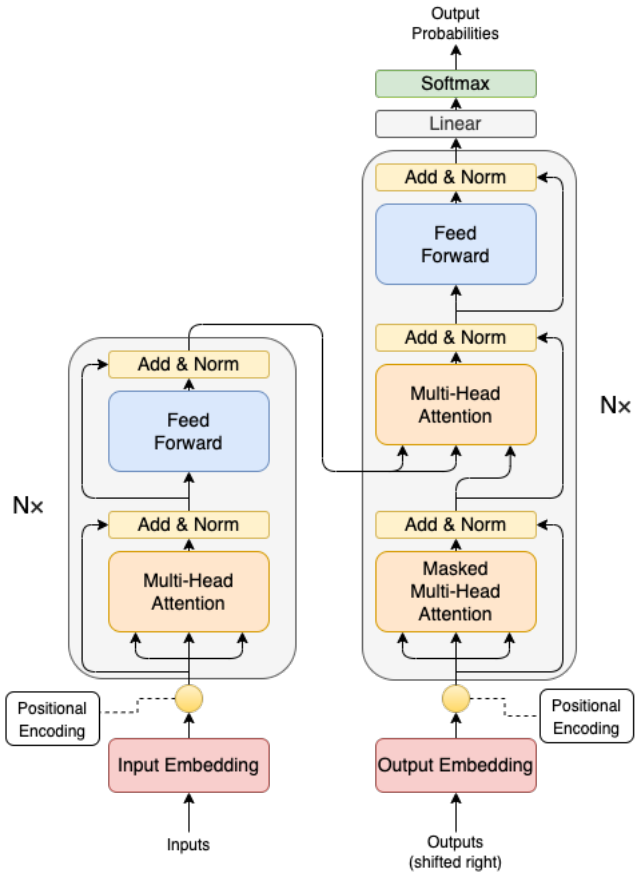

Figure 2.2: *The model architecture of the Transformer. On the left is the encoder and on the right the decoder.*

Decoder Next, we look at the decoder. The architecture of the decoder is similar to the encoder. There will be N decoder layers containing two Multi-Head Attention blocks and one fully connected feed-forward neural network

block, see figure 2.2. Similar to the encoder, there will be residual connections and layer normalization.

As figure 2.2 shows, there has to be an output embedding injected into the decoder. This is because the Transformer decoder is auto-regressive. It means that it consumes previously generated symbols and uses them in the calculations for the following symbol. Also, we can see that the output embedding is passed into a Masked Multi-Head Attention block. It is similar to the other Multi-Head Attention block only with a slight difference in how it works during training of the model[12].

In the beginning, nothing has been generated. Then the output put into the decoder is a start token ”<start>”. This token means that the Transformer has not predicted any words yet. For the example of machine translation, we can imagine having the sentence ”He likes books” as an input into the encoder. The encoder processes the embeddings and creates new vectors. Then, the ”<start>” token is put into the decoder to get the translation. It goes through the Masked Multi-Head Attention. Next, it goes through the Multi-Head Attention together with the encoder vectors. Here is where much of the mapping between the languages happens. The decoder attends to the vectors from the encoder (the English text) and the vectors of what it has already generated (the Swedish text). New vectors are then sent to a feed-forward network. When the vectors have gone through all N decoders layers, they are put through a linear layer and a softmax function which outputs the probabilities of which word should be next. If our model is well-trained, it might output the word ”Han” which is Swedish for ”He”.

The decoder then runs again, but now with two word tokens ”<start> Han”. The decoder stops when the generated output is a token which indicates the end of translation. In our example we hope to get the translation: ”<start> Han gillar böcker <end>”.

2.3.4 BERT language representation model

The Transformer opened up new possibilities. By using it as a building block, researchers at Google created what they called Bidirectional Encoder Representations from Transformers (BERT)[14]. BERT only uses the encoder part of the Transformer previously discussed. BERT is said to be bidirectional because it does not read the text in one direction. It is made like a Transformer and reads the text all at once. So BERT can derive a word’s context from both previous and the following words. Also, the word embeddings that BERT uses are not as simple as we explained above in the Word embeddings section 2.3.1.

The same word can have different word vectors depending on its context. So the word ”dog” in the sentence ”I ate a hot dog” does not get the same word vector as in ”My dog runs so fast!”. Also, the embeddings scheme BERT uses has embeddings for parts of words. Compound words like ”football” would be split up into ”foot” and ”ball”. These word embeddings are called WordPiece embeddings[15]. This type of embedding strategy helps avoid the problem of encountering a rare word that does not have an embedding. The rare word can likely be split up into pieces that have known embeddings.

With all of this said, BERT's most important thing is its two training phases. The Pre-training and the Fine-tuning. In the Pre-training, the machine builds up a model of the language. Furthermore, in the Fine-tuning, we teach the machine to use its model to solve a specific NLP task.

Pre-training BERT In this phase, most of the heavy lifting takes place. Pre-training BERT is much more computationally expensive than the Fine-tuning. However, this phase only has to be done once on the supercomputers at Google. The Pre-training involves two tasks: Masked Language modeling and Next sentence prediction. Masked language modeling revolves around hiding or masking random words and letting the model predict what word should be there. In the Next sentence prediction, the model gets two sentences labeled A and B. It then has to predict if sentence B follows up on A in a text. The model is trained on these two tasks with a massive text dataset, including the English Wikipedia corpus.

Fine-tuning BERT When we have a Pre-trained BERT, the latter phase can be performed. Here the task-specific inputs and outputs are added to the network, and the network is trained end-to-end. We can train the network on various tasks, including text generation, text summarization, sequence classification, and question answering. In this paper, we will fine-tune BERT on a sequence classification task. It means that we have an input sequence of text, which is a combination of a search query and a paragraph. The model will output a corresponding token for every token in the input sequence. It is important to note that a special classification token called the [CLS] token is always put at the beginning of the input. In our case, the output for this [CLS] token will contain the information about the model's prediction if the paragraph is relevant to the search query. The other tokens will not be used in a classification task like the one we are doing, but they can be used in other tasks where tagging specific words with a label are needed.

2.3.5 KB-BERT

Although pre-training a BERT model is very expensive and requires much text, the National Library of Sweden (KB) has not shied away from the challenge. They have released a pre-trained BERT model on Swedish texts[16]. Most of the text is from digitized newspapers, but also government texts, Swedish Wikipedia, and more. In total, their corpus is of the size of 3.5 billion words. It is a similar amount of text to what the original English BERT had, which was 3.3 billion words[14].

2.4 Search ranking with advanced techniques

With the emergence of new NLP models such as BERT, researchers have now tried new approaches to ranking search results. Researchers at New York University created a search ranking algorithm using the aforementioned BM25 model, followed by using a passage re-ranking method with BERT[6]. The passage re-ranking method takes the top ten search results from the BM25 search and runs those documents through BERT. BERT then decides the final rank of the top ten documents. How this is done is explained in the Methods chapter of this report in section 3.2.4.

The researchers showed that this ranking algorithm was better than any other on the TREC CAR evaluation dataset. We want to do a similar thing in this report but in Swedish with a Swedish evaluation dataset and a Swedish pre-trained BERT model. We want to compare a standard BM25 algorithm to another BM25 algorithm that has its top search results re-ranked by BERT.

Chapter 3

Method

This chapter will explain the methods used in the implementation of the project.

3.1 Creation of search evaluation dataset

The first phase of the project was to create a search evaluation dataset. It was done using texts from Swedish Wikipedia. On Wikipedia, there are about 3 million Swedish articles.

3.1.1 Sampling Wikipedia articles

The first thing to do was to find useful Wikipedia articles. Some Wikipedia articles are relatively short, and we want to exclude them from our dataset. The longer articles are the ones we want. Wikipedia articles are always structured in the same way. They have a title, an abstract, and then there are sections. There can even be subsections under each section. The criteria we used was that if an article has two sections in addition to the abstract text, then keep it. We do not consider ”References” sections, ”See also” sections, or similar sections not containing prose.

We built a scraper program to create a list of articles meeting our criteria. The program looked up random Wikipedia articles in Swedish by querying: `https://slumpartikel.toolforge.org/`. The program then checks if the article meets the criteria and saves its path to a text file. Articles that are too short and do not fit the criteria will be discarded. After all the sampling, we ended up with 60,915 articles.

3.1.2 Scraping

The next step in creating the search evaluation dataset was to scrape the text from all the sampled articles. Every Wikipedia article has a title, sections, and often subsections. We can visualize it like the tree in figure 3.1.

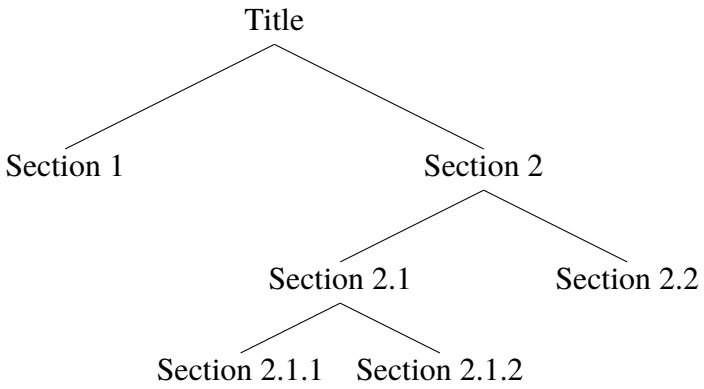

Figure 3.1: *A tree displaying every section in a Wikipedia article*

The paragraphs in an article belong to a node in such a tree. For example, the first paragraph of an article, otherwise known as the abstract, always belongs to the ”Title” node. For each paragraph in an article, we save the paragraph text and the node it belongs to. We save the node on the form [Title/Section 2/Section 2.2], so we practically save all the node's ancestors as well. It will come in handy later when defining the ground truth for different search queries.

3.1.3 Creating an index in Elasticsearch

Elasticsearch is a free search engine based on the Apache Lucene library[7]. Using Elasticsearch, we can create an index for all our paragraphs. We consider each paragraph as a separate document. Therefore, in this report, the terms ”paragraph” and ”documents” will be used interchangeably. In the index created, each document will be on the same form as the document shown in figure 3.2.

In figure 3.2 we have a paragraph from the ”Katt” Wikipedia article. However, the entire paragraph is not shown here, only the first words. This paragraph is under the ”Biologi” section and the second-level section ”Anatomi”. That means the paragraph will get ”Katt” as its first ID, ”Biology”

```
{
    "id1": "Katt",
    "id2": "Biologi",
    "id3": "Anatomi",
    "id4": "",
    "id5": "",
    "paragraph": "Katten har 32 muskler i..."
}
```

Figure 3.2: *One document in the index. One paragraph is one document*

as its second ID, and ”Anatomi” as its third ID. The IDs are all the ancestors of the paragraph in its tree, such as in figure 3.1.

3.1.4 Querying for the ground truth

When creating a search evaluation dataset, we need a search query and a list of ideal search results for that query. With the data previously collected, we can create a list of ideal search results for every node in the tree, such as in figure3.1.

First, a query will be created as a string from the paragraph's section and its ancestors on the form ”$title\ section_1$ ... $section_n$” with a space between every section title. For example from the paragraph in figure 3.2 a search query *”Katt Biologi Anatomi”* can be created.

Next, a list of search results is generated from our indexed Wikipedia corpus. The corpus that we are searching in includes every paragraph in the set of Wikipedia articles. For a query $title/section_1/.../section_n$ a paragraph will get a ground truth relevance score based on the following rules:

- **3 points:** The paragraph is contained in the queried $title/section_1/$ ... $/section_n$.
- **2 points:** The paragraph is contained in a some section placed below the top level section $section_1$.
- **1 point:** The paragraph is contained anywhere in the Wikipedia article.
- **0 points:** The paragraph is not in the Wikipedia article with the title $title$.

By using these attributes, we can rank paragraphs for every query.

3.2 Comparison of search algorithms

The second phase of the project was to fine-tune a pre-trained BERT model. Then, use the fine-tuned BERT as a part of a re-ranking algorithm which we compare to a BM25 ranking algorithm.

3.2.1 KB-BERT

KB-BERT is a pre-trained BERT model. The model has been trained on Swedish text and is public for anyone to use on the Huggingface website[3]. The pre-trained model and the tokenizers for this model were used.

The KB-BERT was loaded with the AutoModelForSequenceClassification model class of the Huggingface Transformers library[17]. This model class has a sequence classification head which puts the output vector corresponding to the [CLS] token discussed in section 2.3.4 through an output layer for classification.

3.2.2 Creating training samples

For the model to solve our task, it needs to be fine-tuned with our data. The model has to do a sequence classification on a sentence pair. It means the model labels a sequence of text with ”relevant” or ”not relevant”. The sequence is two sentences paired together. The first sentence in the sequence is the search query, and the second half is a paragraph. The model then predicts if the paragraph corresponds to the query.

We created both positive and negative training examples. A positive example is created for every paragraph in the corpus by combining the paragraph with the corresponding section string. For example the paragraph in figure 3.2 will be paired with the section string *”Katt Biologi Anatomi”*. A negative example is also created for every paragraph. For a negative sample, the paragraph is paired with a random section string that does not belong to the same Wikipedia article as the paragraph. So for the paragraph in figure 3.2 we could get everything except a string that starts with *”Katt”*.

This sampling method ensures that every positive sample corresponds to a query-paragraph pair with a ground truth relevance score of 3. On the other hand, every negative sample corresponds to a pair with a ground truth relevance score of 0.

Tokenization We tokenized every word with the tokenizer from the National Library of Sweden[3]. Every query-paragraph pair was truncated, so it had at most 512 tokens. If the combined query-paragraph pair is shorter than 512 tokens, we padded the sequence to 512 tokens.

3.2.3 Training

The training was done with 100,000 samples. Half of the samples were positive, and the other half were negative. We used 5% of the samples as an evaluation dataset during the training. So effectively, there were 95.000 samples used for the actual training of the model. The training was done on a cloud GPU and took more than 14 hours.

3.2.4 Algorithm comparison

The final task is the comparison of the two algorithms. The two search algorithms we compared were the following:

- **BM25:** A BM25 algorithm that ranks the documents.
- **BERT re-ranker:** An algorithm that uses BM25 to get a preliminary ranking of documents. Then the trained BERT model comes in and re-ranks the top 10 results. The way it works is that the fine-tuned BERT model, along with its final layer, outputs a probability for how relevant each of the top 10 documents is. Then the documents are exclusively ranked according to these probabilities. So a document with a probability of 0.95 will be ranked above a document with a probability of 0.79.

These two algorithms return the same results from rank 11 and higher. They might only differ in the way the top 10 results are ranked.

It is important to note that in this project, we are only comparing the quality of the search results returned by the algorithms. We will not compare the execution speed of the algorithms, which are vastly different. The BM25 usually returns its results instantaneously. While the BERT re-ranker leads to 10 invocations of BERT, which causes a running time of several seconds depending on the hardware used.

Procedure About 2,000 queries were randomly chosen from a test set of paragraphs. These queries were not used in the training of the BERT algorithm. These are all queries on the form discussed earlier in section

3.1.4. A ground truth ranking was built for all of the queries. The ground truth scores for the paragraphs were used to create an ideal ranking. Then the two algorithms got the chance to return their rankings. The Normalized Discounted Cumulative Gain was then calculated for the BM25 results and the results from the re-ranker.

Chapter 4

Results and Analysis

This chapter will present the findings of comparing the two search algorithms.

4.1 Comparison of the algorithms

BM25 vs. BERT re-ranker Approximately 2,000 randomly selected queries were made using the BM25 algorithm and the re-ranker. The Normalized Discounted Cumulative Gain at rank 10 was calculated and compared for each query. The results of how often each algorithm scored higher can be seen in table 4.1.

BM25 wins	607
BERT re-ranker wins	813
Draws	564
Total searches	1,984

Table 4.1: *How often each algorithm outperformed the other*

The BERT re-ranker returns better search results most of the time. It is also noteworthy to see that the algorithms are often equally good. The BM25 sometimes returns very few relevant documents in the top ten results. When that happens, there are not as many ways to rank the results, and both algorithms might decide to return the same final ranking. Also, in 149 of the 564 draws, the BM25 did not return a single relevant result in its top ten documents. If that happens, the BERT re-ranker, by definition, cannot return any relevant results in the top ten spots. That is an automatic draw between the algorithms.

Average NDCG score Next, let us look at the average Normalized Discounted Cumulative Gain score for each algorithm in table 4.2. From the results in table 4.2 it is evident that the BERT re-ranker improves the NDCG score of the BM25. On average, the re-ranker improves it by 0.007.

Algorithm	Average NDCG
BM25 average NDCG	0.386
BERT re-ranker average NDCG	0.393

Table 4.2: *Average Normalized Discounted Cumulative Gain at rank 10 for both algorithms.*

We can also portray every search sample as a point on a scatter plot. In figure 4.1 we see all the search samples. The samples that form a diagonal line $y = x$ are the samples for which the algorithms produced equal scores. According to table 4.1, there are 564 samples on that line. The samples above the line represent the searches in which the BERT re-ranker produced better results than the BM25. It can be seen that the density of points above the line is a little more than below the line. According to table 4.1, there are 813 points above and 607 points below the line.

Impact of query specificity As explained in the Method chapter, the queries can be of varying specificity. A query could be just the title of the Wikipedia article, like ”Katt”. Alternatively, it could be a string concatenated by the title and the section headers. For example, the query ”Katt Biologi Anatomi” is concatenated from three headers, each of more specificity than the last. We call this a query of level 3, while the ”Katt” query is level 1. The higher the level of the query, the higher the specificity. In table 4.3 the average scores of the algorithms are shown for each query level.

Query level	BM25 NDCG	BERT re-ranker NDCG	quantity
1	0.581593	0.590339	427
2	0.362075	0.367041	1104
3	0.260700	0.272083	405
4	0.271839	0.269672	44
5	0.074612	0.120755	4

Table 4.3: *Average Normalized Discounted Cumulative Gain at rank 10 for both algorithms with respect to query level.*

The BERT Re-ranker has the slight upper hand for all query levels except

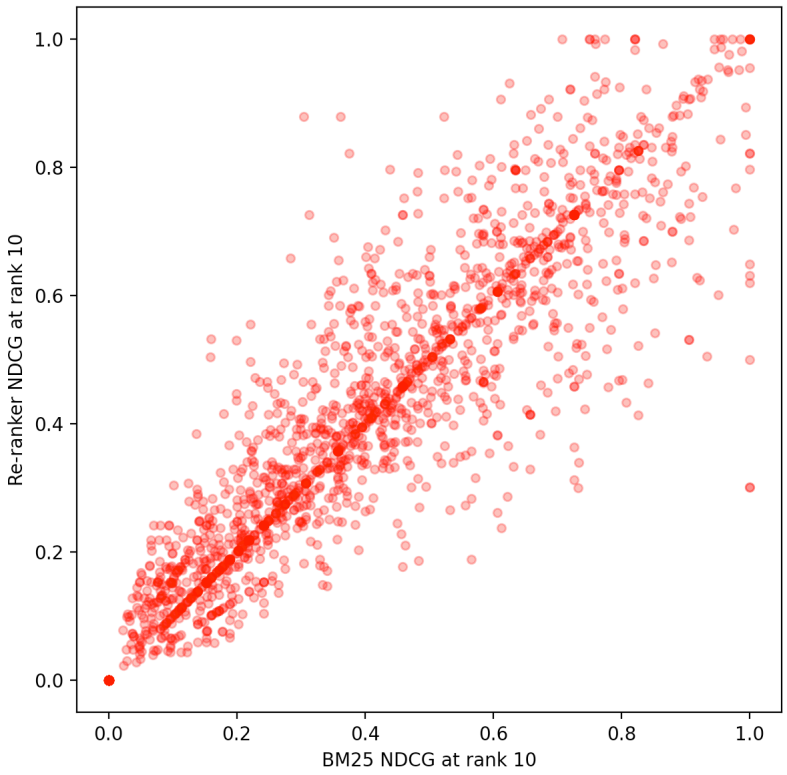

Figure 4.1: *Scatter plot of every search. A single point represents the BM25 score on the x-axis and the BERT re-ranker score on the y-axis.*

for level 4. Then the BM25 has a better score. The small sample size can explain this; only 44 searches had a query of level 4. From table 4.3 it is clear that there is not much difference between the algorithms in how they perform for each query level. The table shows that the BERT re-ranker can outperform the BM25 for specific and broad search queries. However, it is noticeable that the algorithms do a better job with queries of lower specificity.

4.2 Interesting ranking results

BERT re-ranker outperforming BM25 There were a couple of interesting instances where the BERT re-ranker outperformed BM25 by a wide margin. Let us look at an example with the query ”Bokmässa” which means ”Book fair” in English. It is a simple query of level 1. The top ten search results for each algorithm are shown in table 4.4.

For this example in table 4.4, the NDCG at rank 10 was 0.351 for the BM25

Ground truth		
rank	**document ID**	**relevance**
1.	A	3.0
2.	B	1.0
3.	C	1.0
4.	D	1.0
5.	E	1.0
6.	F	1.0
7.	G	1.0
8.	H	1.0
9.	I	1.0
10.	J	1.0

BM25		
rank	**document ID**	**relevance**
1.	K	0.0
2.	F	1.0
3.	L	0.0
4.	M	0.0
5.	H	1.0
6.	N	0.0
7.	G	1.0
8.	A	3.0
9.	O	0.0
10.	P	0.0

BERT re-ranker		
rank	**document ID**	**relevance**
1.	A	3.0
2.	F	1.0
3.	G	1.0
4.	K	0.0
5.	H	1.0
6.	M	0.0
7.	P	0.0
8.	L	0.0
9.	O	0.0
10.	N	0.0

Table 4.4: *The relevance of the top ten search results for the query "Bokmässa"*

and 0.690 for the BERT re-ranker. For the ground truth ranking, the NDCG score is always 1. The BERT re-ranker does a stellar job of re-ranking the results the BM25 gives to it. It almost does the job perfectly. All relevant results are pulled to the top except for the result at rank 5. This example also shows how the re-ranker is dependent on the BM25. The relevant results that fill the top ten in the ground truth ranking were not accessible for the re-ranker. By definition, the re-ranker only looks at the top ten documents from the BM25.

The main reason for the significant difference in the NDCG scores in this example is that the BERT re-ranker was able to put the most relevant result at the top while it was in 8. place in the BM25 results. The text of this document can be seen in figure 4.2.

> En bokmässa är en mässa med inriktning på bokmarknaden, för till exempel förlag, agenter, bokhandel, bibliotek, författare, journalister och läsare. Handel med utgivningsrättigheter är en central del i verksamheten, och inte minst mässans roll som informationskanal.

Figure 4.2: *The text of the most relevant document for the search query "Bokmässa". The BERT re-ranker returned this document as its top result.*

The BM25 algorithm values the quantity of the word's appearances and how short the document is. The BM25 decided the document in figure 4.3 was the most relevant one. It is because it is a relatively short document and the word "bokmässa" appears two times. The same document in figure 4.3 was ranked fourth by the BERT re-ranker.

Åbo bokmässa (finska: Turun kirjamessut) är en bokmässa i Åbo sedan 1990. Den hålls årligen under första helgen i oktober på Åbo mässcentrum.

Figure 4.3: *The text of the document that BM25 returned as its top result for the search query "Bokmässa"*

By pulling the document in figure 4.2 to the top of the rankings, the BERT re-ranker showed it has some understanding of the written text. The document is clearly about book fairs in general, although the word only appears once. The BERT re-ranker can recognize this fact and consequently ranks this document at the top.

BM25 outperforming BERT re-ranker There are also instances where BM25 outperforms the BERT re-ranker. When using the level 2 search query "Microtus socialis Taxonomi" the BM25 showed much better results than the BERT re-ranker. This query is about the taxonomy of a rodent. The top ten search results for each algorithm are shown in table 4.5.

Ground truth		
rank	**document ID**	**relevance**
1.	A	3.0
2.	B	1.0
3.	C	1.0
4.	D	1.0
5.	E	1.0
6.	F	1.0
7.	G	1.0
8.	-	-
9.	-	-
10.	-	-

BM25		
rank	**document ID**	**relevance**
1.	A	3.0
2.	B	1.0
3.	E	1.0
4.	H	0.0
5.	I	0.0
6.	J	0.0
7.	K	0.0
8.	L	0.0
9.	M	0.0
10.	N	0.0

BERT re-ranker		
rank	**document ID**	**relevance**
1.	I	0.0
2.	J	0.0
3.	N	0.0
4.	B	1.0
5.	M	0.0
6.	E	1.0
7.	L	0.0
8.	K	0.0
9.	A	3.0
10.	H	0.0

Table 4.5: *The relevance of the top ten search results for the query "Microtus socialis Taxonomi". There were only 7 relevant documents according to the Ground truth ranking*

For this particular example in table 4.5 the NDCG at rank 10 was 0.733 for the BM25 and 0.300 for the BERT re-ranker. BM25 had the upper hand here but let us look at the documents that ended as the top results for both algorithms. See figures 4.4 and 4.5.

The document in figure 4.4 that BM25 returns as its top result is clearly relevant and is deemed the most relevant document according to the ground truth ranking. However, the documents in figure 4.5 that the BERT re-ranker

Artens taxonomi är något omstridd: Catalogue of Life erkänner inga underarter[5], medan IUCN listar underarterna M. socialis nikolajevi (från Dneprs vänsta bank till Krim) och M. socialis parvus (i (Kaukasus, Stavropol- och Krasnodarområdena samt Dagestan) i Europa. M. socialisgorensis and M. socialis astrachanensis nämns också.[1]

Figure 4.4: *The text of the most relevant document for the search query "Microtus socialis Taxonomi". The BM25 returned this document as its top result.*

Mellansork (Microtus oeconomus) är en art i underfamiljen sorkar.
Microtus pennsylvanicus[2][3][5][6] är en däggdjursart som först beskrevs av George Ord 1815. Microtus pennsylvanicus ingår i släktet åkersorkar och familjen hamsterartade gnagare.[7][8] IUCN kategoriserar arten globalt som livskraftig.[1]

Figure 4.5: *The first two documents that the BERT re-ranker returned as its top results for the search query "Microtus socialis Taxonomi". Both of these results are irrelevant according to the ground truth ranking.*

returns are about the taxonomy of some other species that has a similar name. BERT shows its understanding of the word "Taxonomi" with these two results. The word "Taxonomi" is not explicitly mentioned in the two documents, but they contain information about taxonomy. So the error that the BERT re-ranker makes here is that it fetches the correct information but for the wrong species.

Chapter 5

Discussion

5.1 The BERT re-ranker

The BERT re-ranker proves itself to be a reliable ranker of search results. It can understand the context in Swedish text and rank the search results accordingly more often than not.

The re-ranker was trained on 100,000 training samples. More samples could have been used, and we attempted to use more than 100,000 samples. However, those attempts were not successful in producing a better model due to the GPU units' memory limitations. Training on more samples could have produced a better model and a better BERT re-ranker.

When looking at the search queries where the BM25 does a much better job than the BERT re-ranker, it is often from a Wikipedia page that does not contain prose. For example, a query which ground truth ranking includes paragraphs with many tables. BM25 is better in these circumstances because it only looks at the words themselves. On the other hand, BERT gets more out of looking at the meaning of a sequence of words because it has been trained on running text rather than on tables.

A thing to note about the results of this project is that the BERT re-ranker implemented in this paper is not better than BM25 in all regards. It was only shown that the BERT re-ranker could return more relevant results than BM25. However, BM25 might still be more suitable in practice because of speed or simplicity. The BM25 is much faster than the BERT re-ranker and is also simpler to implement.

Also, it is worth noting that the slight improvement of 0.007 on the average NDCG score might not mean that the BERT re-ranker is better than the BM25. Since the results do not overwhelmingly favor the BERT re-

ranker, there is a risk that BM25 would perform better on another dataset. However, the results in this paper support previous research from Nogueira and Cho[6]. They showed that a BERT re-ranking algorithm could improve a BM25 ranking algorithm and even became state of the art on the English TREC-CAR dataset[9].

5.2 The Search evaluation dataset

In general, it is hard to build a search evaluation dataset. Usually, a lot of manual labeling has to be performed involving human judges reading through swathes of texts and estimating relevancy. However, the method we used here was a little bit different. By creating an automatic ground truth from Wikipedia text, we can get much more data in a very cheap way[9]. No human labor is involved. This method resulted in a vast evaluation dataset containing text from thousands of Wikipedia articles.

Although the dataset's creation was only a preamble, the dataset is a significant byproduct of the project. Others could use it to validate their search algorithms on Swedish text.

5.3 Criticism

One possible source of criticism could be that the KB-BERT from the National Library of Sweden was in part pre-trained on Swedish Wikipedia text. It means that the KB-BERT model could have seen some texts in our test set, although in an unsupervised way. This could have had some effect on the results of this project.

The ground truth from the search evaluation dataset could also be criticized. Although the method used in this report is a very effective one for quickly creating a large search evaluation dataset, it does not necessarily result in a perfect ground truth as humans perceive it. Some of the documents deemed non-relevant by the ground truth ranking can still be relevant search results if we look more closely at the text of those documents.

5.4 Ethics and sustainability

There are always possible ethical issues with training a neural network. When search engines with a neural network component are deployed, there is always a chance that it has been trained on bad data. This could result in biased search

results that could harm certain people. The neural network will mirror the opinions of the data it is trained on, so we must have good quality data.

There are also sustainability concerns over a project like this. Training of a language model is computationally heavy, and people should take care not to waste more time on it than necessary. For this project, a GPU machine was used from Google Cloud[18] and the total emissions were estimated to be 0.871 $kgCO_2e$.

Chapter 6

Conclusions

6.1 Conclusions

In conclusion, the BERT re-ranker did a better job ranking relevant documents in Swedish than the BM25 algorithm. The Swedish KB-BERT can understand the Swedish language to a good enough degree for the BERT re-ranker to consistently pull the most relevant documents to the top of the search results. The BERT re-ranker generally gives high scores to the documents that include text that has a similar meaning to the search query. On the other hand, the BM25 generally gives high scores to documents including the same words as the search query.

The specificity of the search query does not seem to matter for the performance of the BERT re-ranker. The BERT re-ranker does an equally good job on queries searching for very specific information compared to queries with a broad information need.

6.2 Future work

As discussed in section 5.1, the BERT re-ranker might become better by training it with more data. It did take a lot of time to train the BERT re-ranker. Therefore it was not enough time to do hyperparameter optimization. So the tuning of the hyperparameters could result in a better BERT re-ranker.

References

[1] L. Granka, T. Joachims, and G. Gay, "Eye-tracking analysis of user behavior in www-search," *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 04 2004. doi: 10.1145/1008992.1009079 [Page 2.]

[2] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, pp. 333–389, 01 2009. doi: 10.1561/1500000019 [Pages 3 and 5.]

[3] N. L. of Sweden, "Kb/bert-base-swedish-cased," 2021. [Online]. Available: https://huggingface.co/KB/bert-base-swedish-cased [Pages 3, 4, 22, and 23.]

[4] B. V. Kristjánsson, "Eva," https://github.com/bjarkivk/Eva, 2022. [Page 4.]

[5] ——, "search-ranking-comparison," https://github.com/bjarkivk/search-ranking-comparison, 2022. [Page 4.]

[6] R. Nogueira and K. Cho, "Passage re-ranking with bert," 2020. [Pages 4, 18, and 32.]

[7] "Free and open search: The creators of elasticsearch, elk & kibana." [Online]. Available: https://www.elastic.co/ [Pages 4 and 20.]

[8] E. Sormunen, "Liberal relevance criteria of trec -: counting on negligible documents?" in *SIGIR*, 2002, pp. 324–330. [Page 6.]

[9] L. Dietz and B. Gamari, "Trec car: A data set for complex answer retrieval," 2017, version 1.5, 2017. http://trec-car.cs.unh.edu. [Pages 9 and 32.]

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Page 10.]

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. [Page 10.]

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Pages 13, 14, and 16.]

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. [Page 15.]

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Pages 16 and 18.]

[15] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Page 17.]

[16] M. Malmsten, L. Börjeson, and C. Haffenden, "Playing with words at the national library of sweden – making a swedish bert," 2020. [Page 18.]

[17] "Auto classes." [Online]. Available: https://huggingface.co/docs/transformers/model_doc/auto#transformers.AutoModelForSequenceClassification [Page 22.]

[18] "Cloud computing, ready for business." [Online]. Available: https://cloud.google.com/ [Page 33.]

TRITA-EECS-EX- 2022:210

€€€€ For DIVA €€€€

{
"Author1": { ”Last name”: ”Kristjánsson”,
"First name": ”Bjarki Viðar”,
"Local User Id": ”???”,
"E-mail": ”bvkr@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
"Cycle": ”2”,
"Course code”: ”DA231X”,
"Credits": ”30.0”,
"Degree1”: {”Educational program”: ”Master’s Programme, Computer Science, 120 credits”
,”programcode”: ”TCSCM”
,”Degree”: ”Master’s Programme, Computer Science, 120 credits”
,”subjectArea”: ”Computer Science”
},
"Title": {
"Main title”: ”Re-ranking search results with KB-BERT”,
"Language”: ”eng” },
"Alternative title”: {
"Main title”: ”Omrankning av sökresultat med KB-BERT”,
"Language”: ”swe”
},
"Supervisor1”: { ”Last name”: ”Boye”,
"First name”: ”Johan”,
"Local User Id”: ”0000-0003-2600-7668”,
"E-mail”: ”jboye@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
"L2”: ”Computer Science” }
},
"Supervisor2”: { ”Last name”: ”Hartman”,
"First name”: ”Richard”,
"E-mail”: ”richard.hartman@findwise.com”,
"Other organisation”: ”Findwise”
},
"Examiner1”: { ”Last name”: ”Gustafsson”,
"First name”: ”Joakim”,
"Local User Id”: ”0000-0002-0397-6442”,
"E-mail”: ”jkgu@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
"L2”: ”Computer Science” }
},
"Cooperation”: { ”Partner_name”: ”Findwise”},
"National Subject Categories": "10201, 10208",
”Other information”: {”Year”: ”2022”, ”Number of pages”: ”1,38”},
"Series”: { ”Title of series”: ”TRITA-EECS-EX” , ”No. in series”: ”2022:00” },
”Opponents”: { ”Name”: ”Hamza Benjelloun”},
"Presentation”: { ”Date”: ”2022-05-11 13:00”
,”Language”:”eng”
,”Room”: ”via Zoom https://kth-se.zoom.us/my/jocke”
,”Address”: ”Fantum in Lindstedtsvägen 24, Plan 5”
,”City”: ”Stockholm” },
"Number of lang instances”: "2",
"Abstract[eng]”: €€€€

This master thesis aims to determine if a Swedish BERT model can improve a BM25 search by re-ranking the top search results. We compared a standard BM25 search algorithm with a more complex algorithm composed of a BM25 search followed by re-ranking the top 10 results by a BERT model. The BERT model used is KB-BERT, a publicly available neural network model built by the National Library of Sweden. We fine-tuned this model to solve the specific task of evaluating the relevancy of search results.

A new Swedish search evaluation dataset was automatically generated from Wikipedia text to compare the algorithms. The search evaluation dataset is a standalone product and can be beneficial for evaluating other search algorithms on Swedish text in the future.

The comparison of the two algorithms resulted in a slightly better ranking for the BERT re-ranking algorithm. These results align with similar studies using an English BERT and an English search evaluation dataset.

€€€€,
"Keywords[eng]”: €€€€
Natural language processing, Information retrieval, BERT, KB-BERT, Search evaluation €€€€,
"Abstract[swe]”: €€€€

Denna masteruppsats syftar till att avgöra om en svensk BERT-modell kan förbättra en BM25-sökning genom att ranka om de bästa sökresultaten. Vi jämförde en standard BM25-sökalgoritm med en mer komplex algoritm som består av en BM25-sökning följt av omrankning av de 10 bästa resultaten med en

BERT-modell. BERT-modellen som används är KB-BERT, en allmänt tillgänglig neural nätverksmodell byggd av Kungliga biblioteket. Vi finjusterade den här modellen för att lösa den specifika uppgiften att utvärdera sökresultatens relevans.

En ny svensk datamängd för utvärdering av sökresultat genererades automatiskt från Wikipedia-text för att jämföra algoritmerna. Datamängden är en fristående produkt och kan vara till nytta för att utvärdera andra sökalgoritmer på svensk text i framtiden.

Jämförelsen av de två algoritmerna resulterade i en något bättre rankning för BERT-omrankningsalgoritmen. Dessa resultat överensstämmer med liknande studier som använder en engelsk BERT och en engelsk datamängd för utvärdering av sökresultat.

€€€€,
"Keywords[swe]": €€€€
Naturlig språkbehandling, Informationssökning, BERT, KB-BERT, Sökutvärdering €€€€,
}