Abstract

In this project we will use, and compare, term-matching models and deep-neural network models to rank COVID-19 research documents in the CORD-19 dataset (Wang et al. [2020]). The term matching model used is BM25, and the deep neural network is BERT (Devlin et al. [2018]). Recently it has become more and more popular to apply deep neural networks for information retrieval. Models like BERT have outperformed classical models like BM25. Here we try to achieve the same on COVID-19 research articles.

Here we will try to use two different implementations of BERT; the bi-encoder model and the cross-encoder model. The BERT models were fine-tuned on the CORD-19 dataset. Then the models were further trained on title-abstract pairs. Perturbation techniques were applied to the title-abstract pairs to mimic query-document pairs.

The bi-encoder is used in the same way as BM25, it is used for a full-ranking. The results from testing the bi-encoder showed that training the model for the CORD-19 dataset improved the performance. Different pooling methods affected the model, with max-pooling working best. However, when the bi-encoder was compared with BM25 it performed significantly worse. The bi-encoder got a nDCG(10) = 0.561 and the BM25 got a nDCG(10) = 0.658.

The cross-encoder is used for re-ranking the top k results that was retrieved from BM25. When re-ranking the top 10 results the cross-encoder improved the ranking and got a nDCG(10) = 0.713, without the re-ranker the result was nDCG(10) = 0.658. While this seems promising, when changing the number of documents re-ranked the difference is not significant. In addition it is not clear if this training method is easy to replicate for other problems.

Acknowledgements

I would like to tank my advisor for this project Thiago Guerrera Martins.

I also want to give a huge thank to my study partners Olav and Johannes. Together we have worked together on over 20 assignments. They have been a great help through the whole master.

Lastly i would like to thank my girlfriend, she has helped and supported me through all five years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this project we will focus on building and optimizing a search engine. Most people are somewhat familiar with the basics of a search engine and document ranking. If you want to look something you will most likely just "Google" it. Search engines connect a user to websites, documents or other resources. The user gives a query to the search engine and the search engine tries to return documents ranked by the relevance.

There are many ways that the search engine evaluates the relevancy of documents. How recent is the document, how many have already clicked on the document, how many other documents or websites reference this document. Here we are going to focus on natural language processes (NLP) for information retrieval. That means that we are going to operate on written language. One of the most common techniques relies on term matching. This means that the algorithm tries to look for terms in the document that matches the query. These models are very efficient and give great results. Most search engines will use these types of models to some extent. Users often understand this and will construct the search as a series of terms. The problem is that these models cannot understand more complex semantic relationships.

In recent times, deep learning models have been adopted for ranking documents. These models sometimes fall under the umbrella of semantic search. The hope here is that these models can understand more complex relationships and infer intent from the user. Then the user can write the query more like a normal question, and not just a series of terms. One of the most popular language models is BERT, and like many other language models built on the transformer network. This model can be applied for several tasks like question answering, segment summarization, sentiment analysis, machine translation and many other problems. These models have become accessible due to open source toolkits like Pytorch and Tensorflow. Huggingface maintains an easy library specifically for natural language processing. Here one can access a huge amount of pre-trained models. This means that anyone can download and implement state-of-the-art language models, and with a decent GPU one can fine tune these models for a variety of tasks and domains. In this project we are going to use and train these models for information retrieval, and compare these models with term matching models.

The search engine needs a corpus from which we can retrieve documents. In this project we will use COVID-19 research articles. During the pandemic it has been important for policymakers, researchers and many others to have access to relevant and high quality research articles. We have to use a search engine to connect the people with the relevant documents. Here we will try to make a retrieval function that is as effective as possible. To evaluate the search engine we will use a dataset with some ground truth. This consists of a list of query-document pairs with a corresponding relevancy score telling if the document is relevant or not.

Related work

Since BERT was released in 2018 there has been a lot of work applying it to information retrieval. On one of the most important document ranking datasets called MS MARCO the leaderboard (Eder) is filled models using BERT directly, BERT inspired models or BERT in an ensemble of models (Han et al. [2020], Gao et al. [2021]). In this dataset they show clear improvements over models like BM25. Usually BERT is used as a re-ranker. Instead of ranking every document it reorders the top documents from another model, like BM25. This is because BERT is much more computationally expensive.

The TREC-COVID dataset has been out for almost a year at this point. The TREC-CORD challenge was released in 5 rounds, and for each round people competed to make the best ranking model. If we take a look at the fifth and last rounds leader-board we can see that many top spots are taken by models based on BERT (trec covid [2020]). There are also some models based on BM25, but they are not as high up. Several of the models even combine BERT, BM25 and other models into a larger ensemble method. This shows that it is possible to train BERT for this task.

Chapter 2

Data

Here we are going to describe the datasets used in this project. We use two datasets, one is the corpus of documents (CORD-19), and the other is for evaluating the search engine (TREC-COVID). It is important to have high quality datasets to evaluate the ranking models.

2.1 Corpus: Cord-19

In this project we are going to use a dataset called CORD-19 (Wang et al. [2020]). This is a collection of research articles relating to the covid-19 pandemic. This dataset is organized by the Semantic Scholar team at the Allen Institute for AI in partnership with leading research groups. They made this dataset to aid in the fight against the global pandemic and also advance research in natural language processing. The dataset also contains articles that can be relevant to the pandemic, but not directly about covid-19. this includes other pandemics like MERS, SARS or even the Spanish flu, research on vaccines and other treatments. The dataset is updated daily, however the dataset we will be using was downloaded on 1st of February 2021. When referring to the "Full Dataset", this is what is meant.

2.1.1 CORD-19: Dataset Fields

It is important to understand the contents of the dataset. Each document has several data fields. Here is an overview of the most important fields which are relevant to this project.

- **cord uid**: Each CORD-19 paper receive a unique ID. However, the same id can be in multiple rows. This is because the same paper can be gathered from different sources. In this project we have removed these duplicates.
- **SHA1**: This is how we will be able to access a more detailed version of the document.
- **Source x**: A list of sources where this paper is received from. There is at least one source. These sources are MedLine (47 %), WHO (40 %), PMC (37 %), ArXiv (14 %), Elsevier (10 %), MedRxiv (3 %) and BioRxiv (1 %).
- **Title**: The title of the paper
- **Abstract**: The abstract of the paper.
- **Published time**: The publishing date of the paper. This is sometimes just the year or it can be the specific date.
- **Authors**: This is a list of the authors of the paper

- **Body text**: Here is the full text of the paper. This is not always available.

The most relevant fields for the information retrieval algorithms will be the **title**, **abstract** and **body**. These fields are language strings, and natural language processing is the main aim of this project. These fields are written in English, so there will be no issue of different languages. Ideally every document has data in each field, but we are not so lucky. If we look at the most important fields we can see that many documents are these fields. If we want to utilize all these fields we need to know that with most algorithms, documents with all fields will score better.

Field	# Docs with field	% Docs with field
Title	420.112	99.9
Abstract	307.502	73.2
Body	110.240	26.2

The quality of the fields can vary from document to document. There is no way to manually check every field, and the quality. The title is usually good and informative. However the other fields are of varying quality when they are present. Sometimes the abstract is just a copy of the title, sometimes it is just a description like "reply", "Letter to the Editor" or even "No Abstract Available". The same is true for the body text. In total there are about 1700 documents where the abstract is shorter than the title. This is not really a huge problem.

We can also look a bit closer at the publish time of the articles. Here around 78 % of the documents have been published after 2020. Most of the documents published before 2020 are not directly about Covid-19, this is around 1/5 of the documents.

2.2 Evaluation: TREC-COVID

In order to make a good search engine we should be able to evaluate the search engine. This is where the TREC-COVID dataset comes in (Roberts et al. [2020]). TREC-COVID is an ad-hoc dataset, this means that it is not a part of CORD-19, but something added after. This dataset contains 50 different query topics, and each topic has a list of documents with corresponding relevancy scores. The relevancy scores are 0 - not relevant, 1 - partially relevant and 2 - relevant. Each query has around 500 - 2000 evaluations, a more detailed breakdown can be seen in figure 2.1. Here we can see that some topics have several hundred relevant documents, while others have less than 100. This dataset was released in 5 rounds during the summer of 2020.

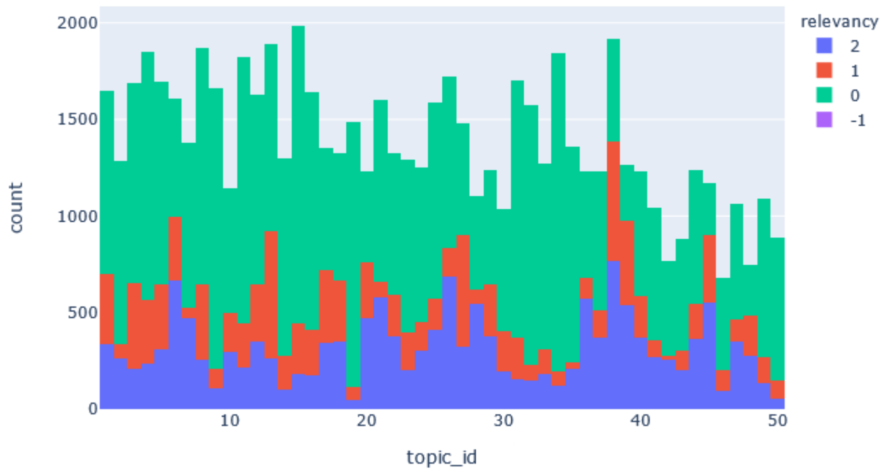

Figure 2.1: Breakdown of evaluations for the different topics. There are a very few documents with relevancy -1, these are removed.

Each topic has a query, question and narrative. The **query** is a short string, this is how we imagine a user will interact with the search engine. The **question** is a more fleshed out version of the query. The **narrative** gives more context on why someone would ask this question. We can look at one example of a topic:

- **Topic id**: 12
- **Query**: how does coronavirus spread
- **Question**: What are the transmission routes of coronavirus?
- **Narrative**: Looking for information on all possible ways to contract COVID-19 from people, animals and objects

For this topic and most other topics the query is clearly much less specific than the question or the narrative. This means that a perfect search engine is impossible because there will be some ambiguity related to the what the query actually want.

For this topic we can also look at some document titles and their relevance score.

Document title	Cord uid	relevancy
An Ounce of Prevention: Coronavirus (COVID-19)...	07l9hqsr	2
Emergency trauma care during the outbreak of c...	05nnihst	1
Infection control practices in facilities for ...	098dcy4z	0

2.2.1 limitations

This is an ad-hoc dataset. There is a possibility of a mismatch between what the evaluation dataset says is a great search engine and what a general user thinks is a good search engine. This means that we can optimize the search engine in ways that are detrimental to the overall user experience. This includes undervaluing functionality that will not be evaluated with this dataset. Another issue is that the queries are made by professionals and everything is proofread, these are not real user cases. Sometimes the user will write queries with grammatical errors or using terms that are not completely correct. Because these are not real user cases, we have to make some assumptions for who the user is and what they want. In the TREC-COVID dataset they imagine that the user is a highly skilled professional, and will know the correct terms used for each query.

Another limitation is that this dataset does not make an evaluation for every document. The full CORD-19 dataset has over 410.000 documents, but only around 37.000 documents have at least one evaluation. Moreover, the last round of TREC-COVID was July 16 2020, but the CORD-19 dataset is updated daily. This means that around 26 % of the documents here were released *after* the last round of TREC-COVID. It is safe to assume that research on different Corona topics is getting better over time. For the documents with no relevancy score we have to make assumptions on the relevancy score. The only feasible way is to assume that these documents are not relevant. The queries in the dataset are very general, at least when it comes to Covid research. So, this assumption is not really safe. For example, if there is an older document with a score of 1, then it is "better" than a newer and better document that is not evaluated. This means that one can improve models, but it will not show up in the evaluation of the model.

In section 2.1.1 we discussed that not every field is available for every document. If we look at the same statistic but only look at the documents with at least one TREC-COVID evaluation we get the results shown in table 2.2.1. Here we can see that the documents in TREC-COVID are much more likely to have an abstract or body. This means that if a ranking model uses the body for ranking, then documents with this field are more likely to be evaluated higher. In the end this will make the problem easier if we can utilise the body or abstract effectively, because documents in TREC-CORD are much more likely to be relevant. The result of this might not be a bad thing for a user, she probably wants documents where all fields are available without passing a pay-wall.

Field	% In TREC-COVID	% In full dataset
Title	100	99.9
Abstract	83.7	73.2
Body	69.9	26.2

We have to make a choice on what documents should be used to evaluate the dataset. Should we only use documents that have at least one relevancy score? Or, should we have a cut off at the last TREC-COVID round? These choices will have an effect on the evaluation. In this project I will use the full dataset as it was downloaded on February 1st 2021. The main reason is that this will make for a more challenging problem by adding a lot more not relevant documents.

To sum up the limitations: The evaluation dataset might not be well designed to reflect how the users will interact with the search engine. Here they have made assumptions on who will use the dataset, and what they want. The majority of the documents have no relevance score for any topic, which means we have to default too the documents being not relevant. There are also some issues with many documents missing important fields.

With all these limitations it is important to remember one thing; these limitations apply to all ranking models. Some models will be more affected than others, and we will have to be careful of this. But, if one model performs much better than another on this dataset, then this will likely be true for a dataset with fewer limitations.

2.2.2 Training data

In this project we are going to use a deep-learning model to rank the documents. These models need to be trained on the domain specific problem, usually they do not work well out-of-box. We need to make a decision on what to use for training data.

The most obvious idea is to use the TREC-COVID dataset for training. There are some issues here. The first is that we need a lot of training data. Here we have 67.000 query-document pairs. This might seem like a lot, but because of how the dataset is built it might not be enough. The model tries to extract a complex relationship between the query and the document. We have a lot of variation in the documents, but not a lot in the queries, there are only 50 different queries.

The next problem is how we are going to split this into training and validation data. One idea is for each query we use a proportion for training and the rest for validation. This is not such a great idea. Mainly because we want to know how the model will work on queries it has not seen. Training on 50 queries and then validating on the same 50 queries might make you believe the model works better than it actually does. The other idea is to split the queries into training and validation. For example, use query 1-35 for training and 36-50 for validation. This has some issues too. Now we only have 15 topics for validation. This really reduces our ability to validate the model thoroughly.

We want to try a different approach. We want to try and generate our own training data, and save TREC-COVID for validation. When TREC-COVID built their dataset many researchers looked through hundreds to thousands of articles. We can not do that. However, we can use some assumptions about the CORD-19 dataset to generate training data. The idea is to use the **title** as a query and the **abstract** as the document. The abstract should be an explanation of the title, this is somewhat the same relationship as we have between a query and the document. It is also safe to assume that if we take the title from one document and the abstract from another, the abstract will not be relevant to that title. Therefore we have a way of generating a lot of training data, both relevant and not-relevant title-abstract pairs.

The title can work as a query, but they are usually not similar. Titles are very specific and grammatically well written. Queries are shorter and more general, and not always well written. Ideally we want the training data to mimic the validation data as close as possible. We will try to do some data augmentation to achieve this, we will come back to the specifics later in section 3.3.2.

The good part about using title and abstract pairs for training is that this method could be used on problems where we have no training data available. It makes the training self-supervising.

Chapter 3

Methods

3.1 Search engine

Here we are going into more detail on how this search engine is structured, how different ranking models work and how we are going to evaluate the models. The main purpose of this project is to evaluate how BERT works for information retrieval. In order to do this we need a benchmark model, and we need to understand how this benchmark model works. Therefore we will also go into how the BM25 model works in some detail.

3.1.1 basic

This is an introduction the basic structure of our retrieval function. We are going to split the retrieval function into three different parts. 1) matching phase, 2) ranking phase and 3) re-ranker. This is shown if figure 3.1.

The matching phase is the first part. The goal of this part is to find documents that are potentially relevant. This function is the only one that runs over every document in the corpus, therefore it is the simplest and fastest.

We can describe the matching function as the following:

$$\text{Matching-function}(Q, D) \rightarrow M,\ M \in \{0, 1\}. \tag{3.1}$$

Here we can see that each document gets ascribed a Boolean value. We want this function to have a high recall (3.2), that means that it is able to retrieve most of the relevant documents. The matching phase should also reduce the corpus for the ranking phase. We want to balance a high recall while retrieving as few documents as possible.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} \tag{3.2}$$

Now we are going to look at the ranking phase. The ranking phase orders the matched documents based on the relevancy. Because the matching phase has downsized the documents we can use more complex functions. This function tries to rank the documents that are relevant to the query at the top. In other words we want to achieve a high precision (3.3).

$$\text{precision(k)} = \frac{|\{\text{relevant documents}\} \cap \{\text{Ranked higher than } k\}|}{k}. \tag{3.3}$$

Unlike the matching phase the ranking function assigns each document a score (3.4). This score indicates how relevant a document is, the higher the score, the more relevant. This value is not that meaning full on its own, but rather can be used to compare documents. If document A has a higher score than document B then it is considered more relevant.

$$\text{Score}(Q, D_i) \rightarrow s_i,\ S \in \mathbb{R}. \tag{3.4}$$

In this project we are going to split the ranking phase further into two parts. Both parts try to achieve the same goal. The difference is that the second ranking phase only ranks the top k documents from the first ranking phase.

In other words, for each step the model complexity increases and the number of documents decreases. In figure 3.1 we have made an example of how this can work. In this figure green are relevant documents and red are not relevant, there is no fuzzy logic. In the first step we run the matching phase. Here we can see that we reduce the number of documents. This is not perfect. Here we want to retrieve all the green documents and none of the red. Here we miss two green documents and retrieve 2 red documents. The next step is the first ranking phase. Here we rank all 6 documents, but two are not shown. The red document is rated too high in this case. Then we run the second ranking phase. In this case we only rank the top 3 documents. Here the model swaps around two documents and achieves a better ranking. This is a simple demonstration, but it shows how we want the search engine to operate.

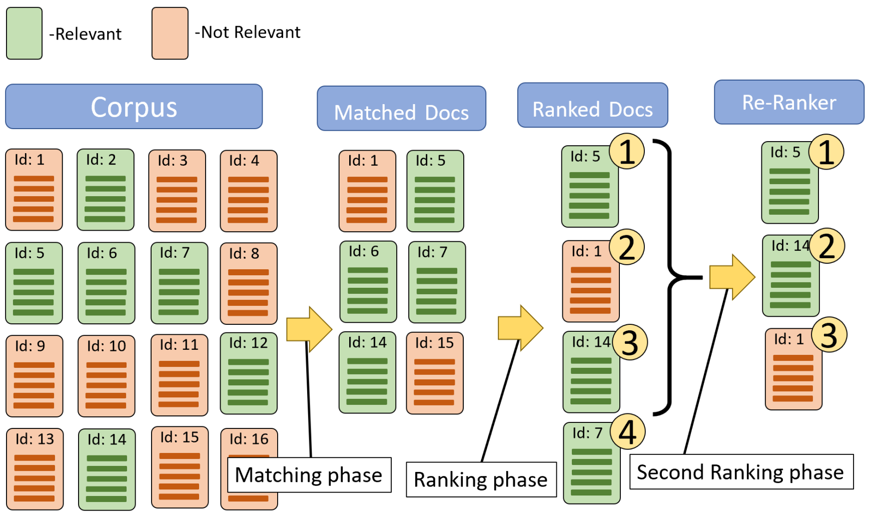

Figure 3.1: Illustration on how the retrieval function works. Relevant documents are green and not relevant red. Here we try to extract the green documents and rank them at the top. In the matching phase we want to retrieve every green document, and as few red as possible. Then the ranking function tries to place the green documents as high as possible.

3.1.2 Term matching VS deep-neural networks

There are two different prominent paradigms we are going to look at. Here we are going to use term-matching models and semantic models. Term matching models build on matching terms in the query with terms in the document. Because of stemming and lemmatization the terms do not have to be exactly the same. For example "infection" and "infectious" could be considered the same term. The problem is that words with similar semantic meaning will not be the same, for example "tall" and "high" are not the same term, but in some contexts they have the same semantic meaning. The term matching models also consider the document as a bag of words. The ordering of the words does not matter. This means that term matching models have a very limited semantic understanding.

The newer paradigm uses deep-neural networks, these can also be called semantic models. Here the hope is that these models can understand more complex semantic relationships. It might read more into the intentions of the user. The issue with these models is that they are large and need a lot of training examples to function optimally. This means that they can be slower than the term matching models, and might not work well out of the box.

3.2 matching phase

The matching phase is trying to retrieve interesting documents. Here we are going to describe some matching functions used in this project. This is not the main interest of this project, but it is good to have some understanding.

3.2.1 Boolean model

The Boolean model is a term-matching model which treats the document and query as a bag-of-words (Habibi Lashkari et al. [2009]). This model treats each term as a Boolean value, either it is present or it is not. If we take a term T and a document D we get that $f(T, D)$ returns 1 if T is in document D and 0 if it is not. This simple function can be used to make matching functions.

The first function is the OR() function. The function is as follows:

$$\mathrm{OR}(Q, D) = \bigcup_{t_i \in Q} f(t_i, D) \tag{3.5}$$

Here Q is the query containing the terms $t_1, t_2 ... t_n$ and D is the document. This means that if the document contains at least 1 term then $\mathrm{OR}(Q, D) = 1$, and is retrieved. The terms in this function will be stemmed. The problem here is that it can be too lenient. If the query is "saklfnjksuef and mlnjlefije" then every document that contains "and" is retrieved. This means that if any of the most common words are in the query (like "a", "and","or"..), then almost every document is retrieved. Then the matching phase is not really doing what it is supposed to do.

A stricter function is the AND() function. This is defined as follows

$$\mathrm{AND}(Q, D) = \bigcap_{t_i \in Q} f(t_i, D) \tag{3.6}$$

Here $\mathrm{AND}(Q, D) = 1$ only if every term in the query is in the document. If at least one term is not in the document, then it will not be retrieved. One problem here is that it can be too strict. If the query contains one word that is misspelled then almost no documents will be retrieved. This is both a good and bad thing. It is good because almost all ranking functions demand that words are spelled correctly, or it will not work as intended. If one term is not correctly spelled then it will not give the results the user wanted. The documents will still be ranked, but not according to what the user intended to write. Having no documents retrieved sends a signal to the user that something is wrong with the query. Most people have experienced results reading "No results matched this query", and then realise that they have made a spelling error.

To summarise AND() will give higher precision than OR(), and OR() will give higher recall than AND(). What function is better depends on the application. Usually the ranking function is better than the matching function at evaluating what documents are relevant, so we want a high recall. The problem is that if we match too many documents then it becomes computationally infeasible to rank quickly. However, this is not a big problem here because we only have 420.000 documents, this is not that many, and we can afford a lenient matching function.

3.2.2 Approximate nearest neighbour (ANN)

Approximate nearest neighbour is a matching function that is similar to k nearest neighbours (kNN). Here we use a model to map the document to a vector $\vec{d}$, the same is done for the query giving the vector $\vec{q}$. The model that does this is often a deep-neural-network. This project uses BERT, but we will come back to this in section (3.3.2). With kNN we retrieve the k documents with the lowest $dist(\vec{q}, \vec{d})$. The problem is that when the dimension of the vectors becomes large kNN becomes slow. In Vespa we can use an approximation of kNN called ANN() (team). Here we aim to retrieve *at least* k documents, but it will not be exactly k. This is a modified implementation of Hierarchical Navigable Small World graphs (HNSW) (Malkov and Yashunin [2016]). Even though this is faster than kNN it is still slower than OR() or AND().

3.3 ranking phase

After the matching phase has found documents that are potentially relevant the ranking function will order the documents depending on relevance. Two Different approaches will be described; term-matching and deep neural network models. For the term-matching we use a version of Term frequency-inverse document frequency called BM25. The deep neural network model used is BERT.

Here we are going one step further by using a two step ranking function. The first ranking function will order every document retrieved by the matching phase, then the next ranking function will reorder the top few results.

Term frequency-inverse document frequency

One of the most powerful term-matching algorithms is called term frequency-inverse document frequency (TF-IDF). This algorithm consists of two statistics. The first statistic measures how frequent a word or term t is in a document D. The other tries to measure how frequent a word is across the corpus. The main idea is the more frequent a term is in a document then the more relevant the document is. However, we do not want the search to be dominated by words like "a", "and", "with", we want rarer words to be weighted higher. The rare words are what really separates documents. Here we are going to break down this algorithm into more detail.

$$\text{TF-IDF}(t, D) = \text{TF}(t, D) \cdot \text{IDF}(t) \tag{3.7}$$

TF - term frequency

The first part we are going to look at is the term frequency. Here we take a term t and a document d and try to measure how frequent a term is in a document. This can be done by counting how many times a term appears in a document.

$$\text{TF}(t, D) = f(t, D) \tag{3.8}$$

The obvious problem here is that this function is biased towards longer documents. One suggestion to account for this is to adjust for the length of the document. This gives the equation

$$\text{TF}(t, D) = \frac{f(t, D)}{\#\text{ Words in document } D}. \tag{3.9}$$

There are a number of other variations. However, most methods use the frequency of the term and accounts for the length of the document.

IDF - inverse document frequency

Every term in a query is not equally important. Let's look at the example "What is the coronavirus death rate?", here "is", "the" or "what" is much less important than "coronavirus", "death" and "rate". The model should know what terms are important and what terms are not. This is done by evaluating how many documents contain a specific term. The fewer documents contain a specific term the better a term is at separating documents. Here we will show one way to define this function

$$IDF(t) = \log\left(\frac{N}{n(t)+1}\right). \tag{3.10}$$

Here we define N as the total number of documents, while $n(t)$ is how many documents contain a specific term t. Here we just add a +1 in the denominator to avoid dividing by zero. It is common to use the logarithm, so the query is not completely dominated by terms that are in very few documents. In summary this function weighs terms that are rare across the corpus higher.

TF-IDF Ranking function

We can use the TF-IDF statistic to build a simple ranking algorithm. Here we take a query Q containing terms q_1, q_2, ... , q_n, and take a document D. This means that we can sum the TF-IDF for each term like this:

$$\text{Score}_{\text{TF-IDF}}(Q, D) = \sum_{i}^{n} TF(q_i, D) \cdot IDF(q_i). \tag{3.11}$$

This means that every document gets assigned a score. This score will be used to determine how relevant a document is to a query. The higher the score the more relevant the document is. One can notice that the longer the query the higher the score. This is not an issue, this is because we only care about the relative score of the documents.

3.3.1 BM25

In this project we will use a state-of-the-art version of TF-IDF called Okapi BM25. Just like normal we take in a query Q containing terms $q_1, q_2, ..., q_n$. This document contains several fields F, in this case this can be the title, abstract or body. Then we can define BM25 like this

$$\text{Score}_{BM25}(Q, F) = \sum_{i}^{n} IDF(q_i) \cdot TF(q_i, F). \tag{3.12}$$

Here we have split the BM25 into the IDF(q_i) and TF(q_i, F). The first part we will look at is the TF, as mentioned this is a measure of how frequent a term is in a field. This is defined as

$$\text{TF}(q_i, F) = \frac{f(q_i, F) \cdot (k_1 + 1)}{f(q_i, F) + k_1 \cdot \left(1 - b + b \cdot \frac{|F|}{\text{avgfl}}\right)} \tag{3.13}$$

Here $f(q_i, F)$ is how often a term is mentioned in a field, $|F|$ is the length of the field, **avgfl** is the average field length of this specific field. The parameters b and k_1 are tuning parameters, the default value of b is 0.75 and the default value of k_1 is 1.2. Here we can see that this value grows with $f(q_i, F)$ and converges towards $(1 + k_1)$.

The next part of the BM25 is the IDF. Here we have defined it as follows

$$\mathrm{IDF}(q_i) = \log\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right). \tag{3.14}$$

Here N is the total number of documents and $n(q_i)$ is the number of fields which have a specific term. We can see that IDF decreases with $n(q_i)$. The fewer the documents that contain these terms the higher the value will be.

Now we want to make this into a ranking function. Here we use documents containing several fields $F_1, F_2, ..., F_m$. Each of these fields are weighed based on how important they are. This gives the final ranking:

$$\mathrm{Score}(Q, D) = \sum_{j}^{m} w_j \cdot \mathrm{Score}_{BM25}(Q, F_j). \tag{3.15}$$

3.3.2 Deep neural networks

Here we will introduce deep neural network models for information retrieval. While term-matching models work well they still they have some issues. One of these issues is miss-matched terms, these could be synonyms like "important" and "essential" or "corona" and "sars-cov-2". The next problem is that term-matching models do not understand more complex semantic relationships. One example of this could be "cases of viruses jumping from animals to humans" and "cases of viruses jumping from humans to animals" are the same in the eyes of BM25. We understand that this is not the same, and we would like for the ranking function to understand this as well. For this purpose we would like to introduce deep neural networks, these models might understand these relationships.

BERT

The model we are going to focus on is called BERT, or Bidirectional Encoder Representation from Transformer. This is a language model developed by Google (Devlin et al. [2018]), and showed state of the art performance on 11 different language tasks, including SQuAD (Rajpurkar et al. [2018]) and GLUE (Wang et al. [2018]). As the name suggests this model uses the transformer network. Often one understands language to be sequential, this has led to language models using recurrent models or convolutions. The transformer network is a much simpler architecture. This is done by only using text that is part of its attention. So, it processes a whole string, and does not use information outside the attention. On machine translation tasks this type of network has shown state of the art performance (Vaswani et al. [2017]), with less training cost and more possibility for parallelization. Here we are going to describe this model in more detail.

The main idea of this model is to train the model on general language tasks. Then, if we want too use the BERT model for another task we do not train the model from scratch, instead one can slightly modify the model and fine-tune it for the new task. This means that once the model is done with pre-training, one can achieve state-of-the-art results by only fine tuning the model. Fine-tuning the model takes much less time than training from scratch The reason this works is because of transfer learning. This is the idea that we can retain some of the "knowledge" gained on the general task and apply it on different tasks. This works because even though we get a new task the general structure of the English language (or any other language) remains the same.

In this project we are going to use the Huggingface (Wolf et al. [2019]) library with Pytorch (Paszke et al. [2019]) to implement and train the BERT models. Also the models are going to be trained in Google Colab on GPUs, usually the training is only takes a few hours.

Tokenizer and text Pre-processing

The BERT model can not operate on unprocessed strings. The string has to be tokenized. The process of tokenization means breaking a sentence into discrete tokens, usually these tokens are words or symbols. The method used by BERT is called word-piece tokenizer (Wu et al. [2016]). There are tens of thousands of words in the English language, and many words have several forms like ”run”, ”running”, ”ran” and so on. This One way to deal with this complexity is to use stemming or lemetazation. This is to return a word to the base form, removing all endings. So, ”jumping” becomes ”jump”. These endings are not trivial, they carry semantic meaning, removing the endings leads to a loss of information. The word-piece model does not remove any endings, they are saved and utilized. Words are split into word-pieces, so ”jumping” will be split into ”jump” and ”##ing”. This has several benefits, one being that we can tokenize many more words. Secondly the endings of words usually carry some meaning which is the same across many words. The ”ing” part of ”jumping” and ”running” has the same meaning.

Using wordpieces solves part of the problem of the large vocabulary. Still, the tokenizer does not have an infinite vocabulary. The vocabulary is created by looking at examples and finding the most frequent words and word-pieces. Because this is generated from examples this works for most languages and contexts. The tokenizer we will use contains 32.000 different tokens.

The model also has some very special tokens, which have important functions in the BERT model. These tokens are; the [CLS] token, this is used for downstream tasks. The [SEP] token separates two sentences. The [PAD] token is just an empty token. Lastly, [UNK] indicates when a word is unknown. We will come back to these tokens later.

After the text string is broken down to discrete tokens we turn the tokens into numerical integers. It is these numerical integers that the model will operate on. In addition to the numerical tokens we will add an **attention mask**, and **token type ids**. This is illustrated in figure 3.2. The attention mask says what part of the tokenized input the BERT model will look at. The input is a fixed length, but the length of the sentence can wary, the attention mask says what part of the input is actually part of the sentence. The segment embeddings are made so we can take in two sentences, the segment embeddings declare what is part of segment A and what is part of segment B. All these parts are combined as the input to the BERT model.

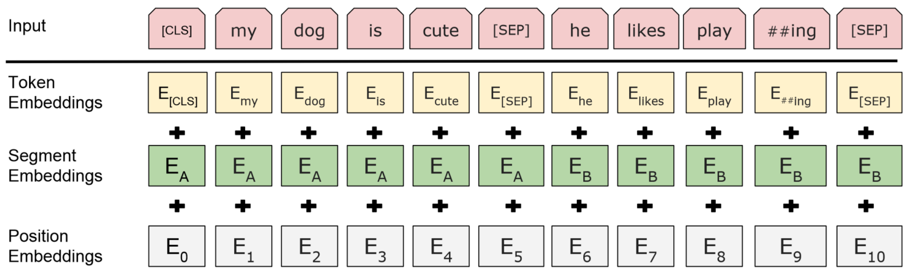
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.2: Illustration from the original BERT paper (Devlin et al. [2018]). This shows the inputs to the BERT model after it is tokenized.

For every application we have to have a specific vocabulary. For example, we can not use an English vocabulary for a French application. It is similar for the CORD-19 dataset. For this task we will increase our vocabulary. This is done by going through the CORD-19 dataset and finding the most frequent words, and then adding the 1000 most frequent words not already in the vocabulary. We can see the result of this in figure 3.3. Here we can see that adding some tokens to the vocabulary reduced the length of the tokenized title from 21 to 10. This is an exaggerated example, but it shows the idea. By adding these 1000 tokens we reduce on average the tokenized length of the title by 11% and abstract by 6 %. This has two advantages; we can fit longer segments into the model and it is easier to learn about ”hydroxychloroquine” when it is represented by 1 token instead of 7.

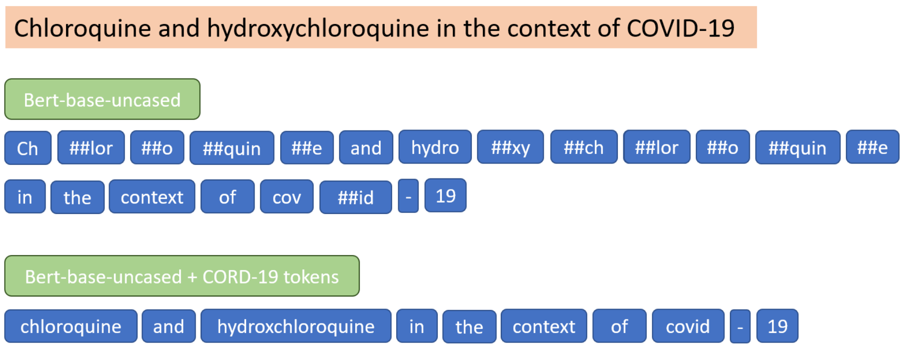
Chloroquine and hydroxychloroquine in the context of COVID-19
Bert-base-uncased
Ch ##lor ##o ##quin ##e and hydro ##xy ##ch ##lor ##o ##quin ##e
in the context of cov ##id - 19
Bert-base-uncased + CORD-19 tokens
chloroquine and hydroxchloroquine in the context of covid - 19

Figure 3.3: One example of a tokenized title. Using bert-base-uncased we get 21 tokens, after adding some CORD-19 specific tokens to the vocabulary it becomes 7. After this the tokens will be turned into integers.

While the word-piece tokenizer is flexible it does have some issues. When doing search engine tasks we often use very specific names, and not general phrases. We want to know some very specific things. Maybe, "AstraZenika vaccine bi effects", it does not help to know something general about vaccine bi-effects. We want the specific effects of the AstraZeneca vaccine. But the vocabulary can not store all names that can be useful for a search engine. The worpiece model can struggle compared to term matching algorithms on very specific terms.

The vocabulary that is relevant for a user can move very rapidly. It only took a couple of tweets from an orange man before everyone wanted to know what hydroxychloroquine was. The vocabulary will need to be continually updated, and then one needs enough examples to train the subsequent model. In practise this is very hard. This issue will not show up here, here we use a very static dataset for validation. In a real application it can move fast. This is not an issue for models like BM25. This model does not really have a limited vocabulary and does not require examples to train on.

model architecture

Here we are going to go more into detail on the structure of the BERT model. This is the part of the model that will stay the same regardless of the downstream tasks. The BERT model can have many different forms, but they use the same building blocks. Here we are going to look at the BERT-BASE model, however models like BERT-LARGE, RoBERTa are very similar.

The base model takes in up to 512 tokens. Then this is passed through 12 transformer and self-attention blocks, this is illustrated in figure 3.4. The hidden size of the token embeddings is $H = 768$. This means that the output embeddings for token t_i has the dimension $t_i \in \mathbb{R}^{768}$ for $i \in 1, 2, .., 512$. These are the vectors we are going to use for downstream tasks.

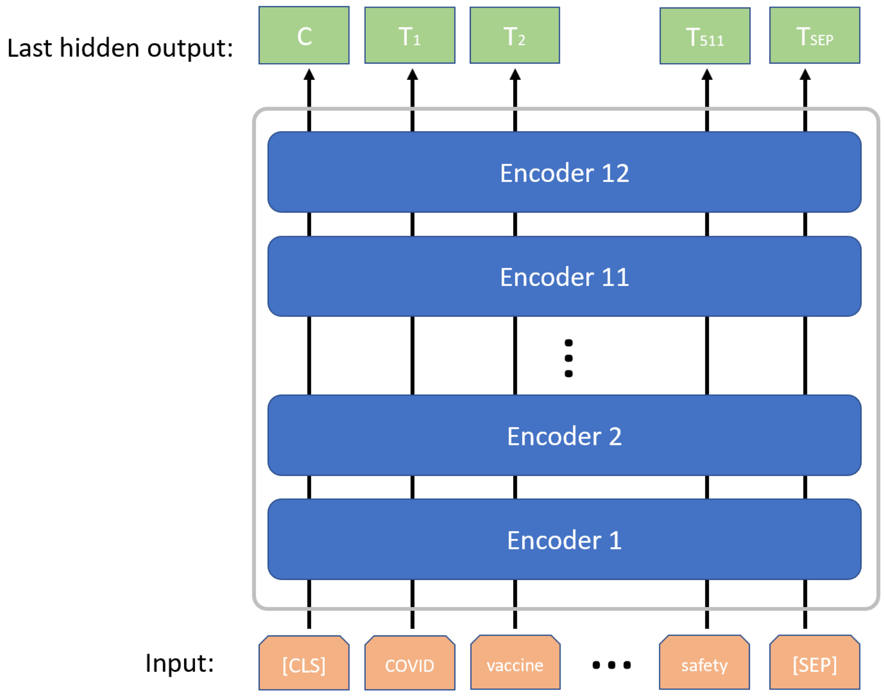

Figure 3.4: This is an illustration of the BERT$_{BASE}$ model. Here we pass in the tokenized input ids, and each token gives a vector as a output.

The training of this model is done in two phases; pre-training and fine-tuning. The pre-training consists of two general language tasks, masked language modeling and next sentence prediction. We will explain these tasks in more detail later. These tasks are self-supervising, meaning it can generate training examples by itself from a corpus. The dataset used for the model is the BooksCorpus (Zhu et al. [2015]) and the English Wikipedia. These datasets contain long uninterrupted sequences. After the pre-training we have a general language model, this can be fine-tuned for specific tasks. Here the specific task is information retrieval. When applying this model to other tasks extra inference layers are added. Because these layers can be small, the fine-tuning of the model is much quicker than the pre-training.

Masked Language Model - MLM

One of the pre-training tasks is the masked language model. In this task the model tries to guess a masked word in a string. One example of this task could be "Guess the [MASK] token in these sentences.", here the mask would be "masked". During training we swap 15 % of the tokens with a [MASK] token. The general idea is that if a model is able to guess the masked token often then the model has some general language understanding.

The difference between BERT and many other language models is that this task is solved *bidirectionally* instead of from left-to-right. This means that we use the words to the left of the masked token and to the right. The left-to-right models only use the left part. In the BERT paper the bidirectional method worked better than the left-to-right method.

Here we will show the BERT model in action on the MLM task. Let's take the phrase from the Wikipedia page for Norway:

- "The southern and western parts of Norway, fully exposed to Atlantic **storm**$_1$ fronts, experience more precipitation and have milder **winters**$_2$ than the eastern and far **northern**$_3$ parts. Areas to the east of the coastal mountains are in a rain shadow, and have lower rain and snow totals than the west."

Here I have marked three words in bold. If we mask these words we can see what words the model guesses. The results from this are shown in table **??**. The model manages to guess at least two of the masked words, and it gets the third token correct on the fourth guess. If we take a look at the incorrect guesses we can see that they are reasonable given the context. Clearly this model has some language understanding.

	storm$_1$		**winters**$_2$		**northern**$_3$	
Rank	**token**	score	**token**	score	**token**	score
1	weather	0.49	**winters**	0.77	**northern**	0.44
2	cold	0.15	temperatures	0.11	eastern	0.29
3	ocean	0.12	summers	0.04	western	0.09
4	**storm**	0.11	climate	0.01	southern	0.08

Table 3.1: This is the result from the MLM task. Here we can see the correct token, the guessed token and the corresponding confidence score. Here we can see that it guesses two of the words correctly. One can also see that the incorrect guesses are reasonable.

The example above was a phrase from a Wikipedia page, this model was trained on Wikipedia pages. It is reasonable that it will perform well here. However we are going to use this model for COVID-19 research documents. This is a highly specialised domain. In this dataset "corona" is the 40th most frequent word. So, if we give a domain specific task to the base model, we might not get such great results. If we give the phrase "Covid19 is a [MASK]". Then the base model will give these results; protein (0.77), gene (0.04), receptor (0.02) and so on. Not bad guesses if you have never heard the word before, but with a little fine tuning on the dataset we can do much better. After all, the research documents are still written in English, so the model only has to learn some new words. After training the model on the CORD-19 dataset we can try the same phrase. The results from the phrase "Covid19 is a [MASK]" are now; pandemic (0.39), disease (0.05), hoax (0.02). This is much better, however we could do without the hoax! If we were to change the phrase too "Covid19 is *an* [MASK]" the first result is **epidemic**, showing that the model recognises some grammar. Even if the model has low confidence in the token "hoax", this still demonstrates one of the issues with this type of training. The model learns patterns in the training data. This means that if the training data is biased then the model will be biased as well. It is very important that the training corpus is of high quality.

Next Sentence prediction - NSP

The second task BERT is pre-trained on is the Next Sentence Prediction task. In this task we have two sentences, A and B. Sometimes the sentence B follows A and sometimes not. The model tries to correctly identify when this is the case. One example of this could be:

- A: "She walked to the store", B: "There she bought an ice cream"
- A: "He took the bus downtown", B: "Norway has a total area of 385,207 square kilometres"

In the first case the sentence B does follow A, but this is not true for the second example. When training it is common to make True examples 50 % of the time, and false 50 %. To solve this task we do not change the architecture much. We only add a logistic regression model on the **CLS** embedding. This means that we get a probability in the end, so the model gives a probability that sentence B follows A.

In the original BERT paper they showed that this task improved performance on downstream tasks. However, in a more recent paper it was suggested that this task does not really improve the performance (Liu et al. [2019]). Here they use the same model architecture as BERT$_{\text{LARGE}}$, but they do not train on the NSP task. In this paper it is suggested that by using a more robust approach to the MLM task you can improve the model on downstream tasks, and using the NSP task does not really help. Here we will not use NSP as a part of the pre-training.

Even if we are not using NSP for pre-training, however this task is very similar to the task we want to achieve. Instead of using two consecutive sentences we want to know if the document follows the query. This is illustrated in figure 3.5. All we really need to add on the BERT architecture is a logistic regressing on the [CLS] embedding.

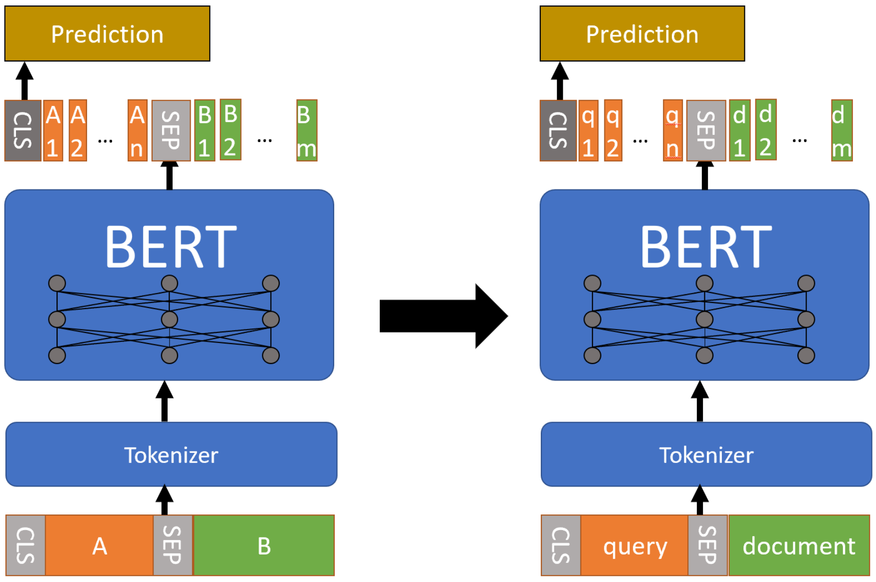

Figure 3.5: This shows how we use the NSP-task as an analogue for information retrieval tasks.

For information retrieval

There are several ways we can use BERT for information retrieval. The two main ideas are illustrated in figure 3.6. Method 1) embeds the document and query separately. Then we add an ad-hoc model to make a ranking. This is also called the **bi-encoding** model. The other model takes the query and document as input together and embedded it together, this is called **cross-encoding**. Both methods have advantages and disadvantages. The bi-encoding model can embed the document before the query arrives, then we only need to embed the query and run the ad-hoc model. This is much faster than method 2). On the other hand method 2) will be able to make better predictions, provided we have good training data.

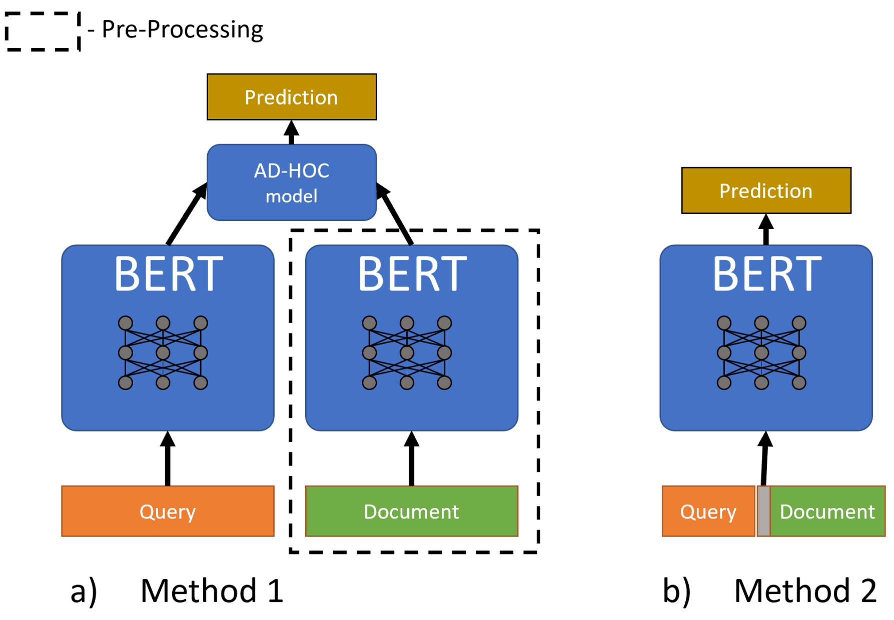

Figure 3.6: This shows the two approaches to the information retrieval using BERT. On the left we have the bi-encoder model, and on the right is the cross-encoder.

Method 1): bi-encoding

Firstly we are going to look at method 1. Here we are going to use the BERT model to make an embedding of a text string. This means that we take the query and make an embedding $f_{\text{BERT}}(q) = \vec{q}$. Likewise this is done for each document $f_{\text{BERT}}(d_i) = \vec{d_i}$. The document embedding is done in the preprocessing which saves a lot of time. These vectors are used by a model and give a score $f_{\text{ad-hoc}}(q, d) = s$.

Here we are going to use the simplest possible model, the cosine similarity. This is defined as:

$$\text{cos-sim}(\vec{q}, \vec{d_i}) = \frac{\vec{q} \cdot \vec{d_i}^T}{||\vec{q}|| \cdot ||\vec{d_i}||} \tag{3.16}$$

Here we say that the cosine similarity is the relevancy. This method requires no tuning of the ad-hoc model. This method will also be used as a matching method, see section 3.2.2 for more detail.

There are several ways to make these vectors using BERT. Here we are going to try 3 different pooling strategies.

- **CLS**: This is the "normal" pooling method. Takes the output from the [CLS] token.
- **MAX-pooling**: Each token will have an output embedding $t_i \in \mathbb{R}^{768}$. So, if the input has n tokens, then the max-polling becomes: $\text{polling}^j_{max} = \max(t^j_1, t^j_2, ..., t^j_n)$ for $j = 1, 2...768$.
- **Average-pooling**: Here we use the same approach as the max-pooling, but instead the formula becomes $\text{polling}^j_{average} = \text{average}(t^j_1, t^j_2, ..., t^j_n)$.

In addition to these pooling strategies we are going to look at what layer we are going to take the embeddings from. If we look at figure 3.4 we can see that we get outputs from all 12 encoder layers. Normally we take the outputs from the last encoding layer, but this might not be the best method. Here we will test if it is better to use the outputs from an earlier layer.

Because of the limited number of tokens that can fit into the BERT model we have to sometimes truncate the document string. For the title this is never an issue, this sequence is never over 512 tokens. This can be an issue for the abstract. Here we will truncate the end of the string, keeping the first part.

Method 2): Cross-encoding

Now we are going to move over to method 2). Like we mentioned in 3.3.2 we are going to use the architecture from the next sentence prediction task. The issue here is what to use for training data. In the TREC-COVID dataset we have almost 69.000 query-document pairs, but the issue is that we only have 50 different queries. For different reasons explained in section 2.2.2, we are reluctant to use this dataset for training. Instead of using this dataset we are going to generate a dataset from the CORD-19 dataset. This is done by using the title as a query and the abstract as a document. Here we can make many more different query-document pairs without using our evaluation dataset.

Just like with the bi-encoder we have a limited number of tokens. Here we will dedicate 62 tokens to the query and 447 tokens for the document. There is usually not an issue with having to truncate the query, 62 tokens can be many sentences, longer than most queries. The document will have to be truncated, here we will keep the first part of the string.

This dataset is not perfect for information retrieval training. Because the title and abstract are written by the same author(s) they are often a perfect match, this is not the case for query-document pairs. Here we are going to try some data perturbation techniques. They are described as follows:

- **Remove Stop Words**: Sometimes we are going to remove the stop words from the title. This is to mimic how users write queries. They are not always written in full sentences, but rather as keywords. A list of all the stop words is shown in the appendix A.3.
- **Short sequence**: We are going to shorten the length of the title and/or abstract. We will always keep a continuous string, but what part of the string is random. This means that we might not get a perfect match between the title and the abstract.
- **Shuffle title**: Shuffle the tokens of the title. The idea is that users do not always write the keywords in the correct order.

We do all this perturbation to mimic the task of document ranking. For each of these perturbations we can add a probability that this will happen and we can change the degree of perturbation. In the results we are going to see if this actually improves the model for information retrieval.

The proportion of true and false title-abstract pairs can also make an impact. Usually we use a 50-50 split, but this might not be an optimal choice. Therefore we will test what the optimal split is.

In addition to the perturbation we will use title-title pairs 10 % of the time, and we will swap the title and abstract 10 % of the time. This is done so we don't always train on abstracts, after all we will also use the title for the retrieval. If the title is never used as a document sting, then titles will not be valued highly.

There are some issues with this training. One of the goals of training a deep learning model for information retrieval is that the model will understand the user's intent better. In this case we will use title-abstract pairs, and we expect that both will use similar terms. This can miss the point leading to the model learning to match terms more than anything else.

This method 2) is much more computationally expensive than BM25. Therefore we can not use it for a full ranking, we will only use this model as a re-ranker for the top k documents. Method 1) is more feasible as a full ranking method, this is because we only use the BERT model once for the query, and the document embedding is done in the pre-processing.

3.4 Evaluation

When creating any model it is important to evaluate the result. Information retrieval uses some very specific metrics. This is usually because we mostly care about the top few results, the user will only look at the top results. Getting these documents correctly is very important. In this segment we are going to introduce the metrics we use to evaluate the models.

We also need a dataset to run the evaluation, this comes in the form of TREC-COVID as described in 2.2. In this dataset we have 50 queries, and unless otherwise mentioned the evaluation is done on all 50 queries. Each evaluation metric is usually calculated for one query at a time. Then we take the average of all the 50 queries.

3.4.1 Match Ratio

It is often interesting to know how many documents the matching phase retrieves. A simple metric we can use is the match ratio described as

$$\text{MatchRatio} = \frac{|\{\text{retrieved documents}\}|}{|\{\text{all documents}\}|}. \tag{3.17}$$

This number will not tell you much on its own. If we have very complicated ranking functions and a large corpus this number needs to be low. If we have a cheap ranking function, like BM25, and a small corpus then this number can be close to 1.

In this instance we do not have very many documents, only around 420.000. This means that we will not focus too much on this metric.

3.4.2 Precision

The precision gives the proportion of retrieved documents that are retrieved. We define this as

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}. \tag{3.18}$$

Usually we want this value to be as high as possible. This metric can be used to evaluate both the matching phase and ranking phase. However, when evaluating the ranking phase we usually care about the top k results. We can modify this metric to only look at the top k results. This can be defined as

$$\text{precision}(k) = \frac{|\{\text{relevant documents}\} \cap \{\text{documents ranked higher than } k\}|}{k}. \tag{3.19}$$

Here we also want a value as close to 1 as possible. This is always possible. If there are less than k relevant documents we can not get a precision$(k) = 1$.

3.4.3 Recall

The recall is the proportion of the relevant documents actually being retrieved. Here we defied it as

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}. \tag{3.20}$$

This metric ranges from 0 to 1, and the closer to 1 the better. This is the proportion of relevant documents we are able to retrieve. This can also be used to evaluate the matching and ranking phase. When applying this to the ranking phase we are only going to focus on the top k results. So, we define it as

$$\text{recall}(k) = \frac{|\{\text{relevant documents}\} \cap \{\text{ranked higher that } k\}|}{|\{\text{relevant documents}\}|}. \tag{3.21}$$

Here we also want the highest number possible, but it is not always possible if there are less than k relevant documents. In most cases here we are going to use $k = 10$. In TREC-COVID there are more than 10 relevant documents for each query.

3.4.4 Reciprocal rank

Usually when ranking documents the first document is much more important than the 10th document. Here we will introduce a metric that tries to capture this. This metric is the reciprocal rank. Here we take the rank of the first relevant document. This means that if the first relevant document is in the third spot then the rank is 3. The score we use is 1 divided by this rank. If the first relevant document is in the third spot then the reciprocal rank is 1/3.

Here we are going to use this across several queries and take the average. For each query q_i we get a corresponding rank_i. Then we can take the mean reciprocal rank defined as

$$\text{MRR} = \frac{1}{n}\sum_{i}^{n} \frac{1}{\text{rank}_i}. \tag{3.22}$$

If the first relevant document has a rank=1 is twice as "good" as if the rank=2, and tree time as good as if rank=3.

3.4.5 Normalized Discounted Cumulative Gain (nDCG)

There are many good things about the mean reciprocal rank, but there are some issues. MRR does only use the first relevant document, not the full list. This means that if the first and second document is relevant this scores just as well as if only the first document is relevant. Another problem is that the relevance score is binary, either it is relevant or it is not. In the TREC-COVID dataset we have 3 relevant scores (0,1 and 2), so we want to use that as well. For this purpose we introduce the discounted cumulative gain (Lin et al. [2020] s. 25). This metric is defined as

$$\text{DCG}(k, q) = \sum_{i=1}^{k} \frac{2^{\text{rel}(q,d_i)} - 1}{\log_2(i+1)}. \tag{3.23}$$

Here we only look at the top k results from the ranking, and i indicates the rank of the document. $rel(q, d_i)$ shows how relevant a document is to a query. This means that we are able to use all top k documents and utilize the non-binary relevance score.

To make this metric easier to interpret across different queries we want to normalize the score. Therefore we introduce the *normalized* Discounted Cumulative Gain.

$$\text{nDCG}(k, q) = \frac{\text{DCG}(k, q)}{\text{IDCG}(k, q)} \tag{3.24}$$

The $DCG(k, q)$ is as defined above, the $\text{IDCG}(k, q)$ is the ideal Discounted Cumulative Gain. This is the best score possible. In our case it means first comes every document with a relevancy score

2, then every with relevancy score 1 until we reach k documents. This means that the nDCG rankes from 0 to 1, where 1 is perfect ranking. This will be done for every topic, then we can get the average from all 50 queries.

For all these reasons we will put the most weight on the nDCG when evaluating a ranking model.

3.4.6 ROC and AUC

Information retrieval can also be viewed as a classification problem. For each query-document pair we want to classify it as relevant or not relevant. In the document ranking task we usually only care about the documents we are most certain are relevant. Here we are going to use the ROC and AUC to evaluate BERT models. When we use the BERT models we get a score from 0 to 1. The closer to 1 the more certain the model is that the document is relevant. In the classification problem we will choose a confidence threshold t such that if $f(q,d) > t$ then the document is classified as relevant, and if $f(q,d) \leq t$ then it is not relevant. After predicting the class we can place it in the table 3.2 , this is called a confusion matrix. Here we want all documents to be placed in either the true positive or true negative, this means that the prediction was correct.

	Predicted Relevant	Predicted Not relevant
Relevant	True Positive	False Negative
Not Relevant	False Positive	True Negative

Table 3.2: Confusion matrix.

After placing the documents in the confusion matrix we can calculate the true positive rate. The formula for this is

$$\text{True positive Rate} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}. \tag{3.25}$$

This is in other words the proportion of relevant documents that got classified as relevant. This is also referred to the sensitivity. We have a similar statistic called the false positive rate. This is defined as

$$\text{False Positive Rate} = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}}. \tag{3.26}$$

This is the proportion of not relevant documents that get rated relevant. In other words in the classification problem we want a high true positive rate and a low false positive rate.

Until now we have had a constant confidence threshold t. Instead we are going to use a moving threshold from 0 to 1. For each value of t we get a corresponding true positive rate and false positive rate. Then we can make the receiver operating characteristic curve. This is shown in figure 3.7. Here we see that we can find the area under the ROC curve, this is called the AUC. We are going to use the AUC for evaluating BERT-models. When we do this evaluation we only use documents and evaluations in the TREC-COVID dataset. Because the AUC only assumes 2 classes we will only consider whether the documents are relevant or not.

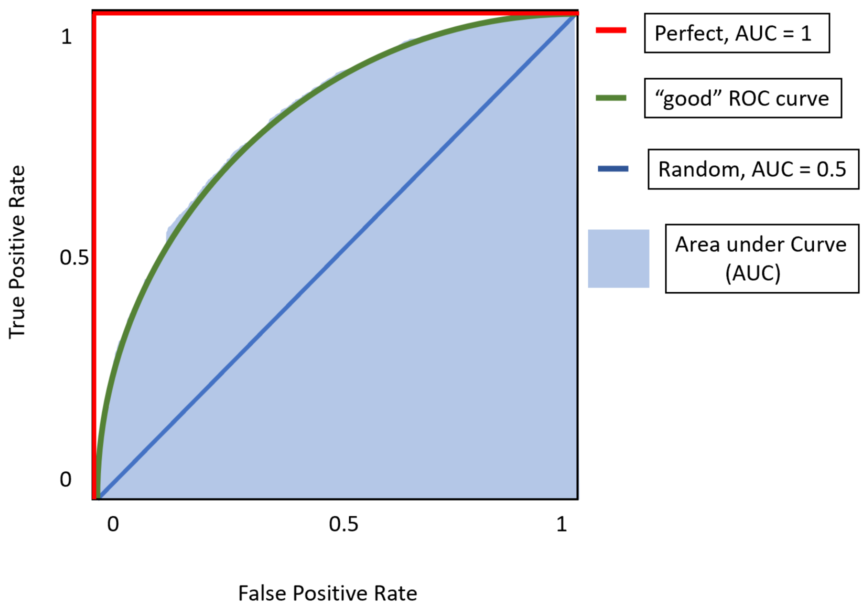

Figure 3.7: Illustration of the ROC. The red line is a perfect classification. The blue curve is a random classifier. The green line is a good model, and the light blue area is the AUC.

The regular metrics like nDCG and MRR are still preferable, but using AUC allows for more rapid testing of the models. This will guide our choice for what model to use for the ranking. In the result section we will be clear about when we look at the classification problem and when we look at the ranking problem, the latter is the problem we care the most about.

Chapter 4

Results and discussion

Here we will test all the different models explained in section 3 for ranking the CORD-19 dataset with TREC-COVID evaluations. The search engine is implemented using an implementation of Vespa (team Vespa) for python called pyvespa (pyv). All BERT based models are trained in Google Colab using V100 GPUs, the training takes about 1-3 hours.

4.1 BM25 Results

The first method we are going to use to rank the documents is BM25(). For this test we will use OR() as a matching phase. The goal of this test is what fields are most valuable and to see if we can improve the ranking by combining fields. The results from this test are shown in the table 4.1. By just looking at the fields individually we can see that the **body** gives the best result. We should be a bit sceptical of this result, in section 4.1 we saw that having a body was much more common in the TREC-COVID dataset than in the full corpus. That means that using BM25(body) can work as a filter for documents in TREC-COVID, and if a document is not in TREC-COVID it is automatically assumed to be not relevant. Therefore, any function that filters out documents not in TREC-COVID will have an advantage.

Now we can take a look at combining different fields. Here we see that combining fields gives the best results, especially when we use different weighting for each field. The weight of the field is not directly the same as how important it is. This is because the BM25 gives different values for each field, although it is only slightly better than BM25(body), and it could just be due to chance. If we look at the top 10 results and take the average of BM25(field) we get 5.0 for the title, 10.0 for the abstract and 7.6 for the body. This gives the weighted mean of 1.3 for the title, 6.0 for the abstract and 1.1 for the body. This model is even more skewed towards the abstract than it seems from the first look. It might be possible to optimize the weighting of the fields even more, but this will work well as a baseline.

Model	Recall(10)	nDCG(10)	MRR(10)
bm25(title)	3.31E-3	0.579	0.699
bm25(abstract)	3.10E-3	0.551	0.631
bm25(body)	4.13E-3	0.655	**0.783**
bm25(title) + bm25(abstract)	3.17E-3	0.570	0.652
bm25(t)+bm25(a)+bm25(b)	4.41E-3	0.649	0.700
0.25bm25(t)+0.6bm25(a)+0.15bm25(b)	**4.52E-3**	**0.658**	0.748

Table 4.1: This shows how one can use and combine different fields for the ranking function using BM25.

Later we are going to look at re-ranking the results from BM25. The main model we will use is the BM25 using all fields with different weighting, we call this model BM25w. We can test this model a bit more. Here we will use the different matching phases OR() and AND(). We will also test the different query fields using the **query** and the **question**. The results from this test can be seen in the table 4.2. We can see that using the question instead of the query gives a big improvement to the model. This is because the question contains more important keywords than the query. However, this is reversed when we use the AND() matching function. Using AND()+question gives the worst result of the models tested. This is likely because the matching function ends up being too strict and excludes relevant documents. If we look at the Match-Ratio only .7 % of the documents are retrieved from the matching phase.

There is one strange result that needs to be explained, namely that AND() works better than OR() when using the query. Because the ranking function is much more complex than the marching function we assume that we want to retrieve as many documents as possible and let the ranking function make the call. What happens here is a weakness of how the nDCG is computed. The matching function changes the ideal DCG, this leads to the AND() giving a *lower* ideal DCG than the OR(). The DCG score for OR()+BM25w is actually higher (DCG(10) = 3.456) than it is for AND()+BM25w (DCG(10) = 3.454). Therefore we will still say that OR()+BM25w is the best retrieval function.

Matching phase	Model	query field	Match Ratio	Recall(10)	nDCG(10)	MRR(10)
OR()	BM25w	query	0.686	**4.52E-3**	0.658	0.748
OR()	BM25w	question	0.958	4.48E-3	**0.722**	**0.775**
AND()	BM25w	query	4.52E-2	4.33E-3	0.679	0.732
AND()	BM25w	question	6.99E-3	3.50E-3	0.579	0.695

Table 4.2: This shows how different matching functions and query fields affect the BM25w ranking.

4.2 Ranking with BERT

Here we are going to look at the results for the BERT models. These models are trained on two different tasks; the masked language model using the CORD19 dataset and the next sentence prediction task with title-abstract pairs. The testing of the models is done using the TREC-COVID evaluations, but this dataset is not used for training. We are going to use the two methods described in section 3.3.2; the bi-encoder model and the cross-encoder.

4.2.1 Method 1): Cos-Sim

The first model we are going to use for ranking the documents is using the cos-sim model explained in 3.3.2. Here we make embeddings for the query and for the document, then look at the cos-sim between the vectors.

One of the things we can test is what pooling method is the best. Here we will also test three models; "Bert-base-uncased", "cord19-bert-mlm" and "cord19-bert-nsp". "Bert-base-uncased" is the original model released by Google, it has not seen a single sentence about Covid before. "cord19-bert-mlm" is trained on the MLM task using the CORD-19 dataset. Lastly "cord19-bert-nsp", this model is further trained on the downstream task of title-abstract pairs. Here we will also test using different fields as the document string. The results for this are shown in the table 4.3. The first thing to notice here is that the title is consequently better than the abstract. This might be because longer strings contain too much information to make a simple vector. If we compare the different models we can see that "bert-base-uncased" has the worst result, then we get "cord19-bert-mlm" and on top we find "cord19-bert-nsp". The model with the best results is the "cord19-bert-nsp" with the max-pooling. This model works worse than "bert-base-uncased" and "cord19-bert-mlm" when using the abstract. However, if we look at the results from BM25 in table 4.1 we can see that the best performing field tends to be the most important. Therefore we will move forward with the "cord19-bert-nsp" with max-pooling, because this is the best overall model.

Model	Document Field	Pooling method max	(AUC) cls	mean
"bert-base-uncased"	both	0.552	0.522	0.5240
	title	0.596	0.562	0.558
	abstract	0.563	0.522	0.537
"cord19-bert-mlm"	both	0.584	0.552	0.528
	title	0.609	0.619	0.575
	abstract	0.569	0.572	0.519
"cord19-bert-nsp"	both	0.567	0.5305	0.527
	title	**0.665**	0.591	0.585
	abstract	0.542	0.522	0.536

Table 4.3: This shows the AUC for different BERT models and pooling methods.

The next thing we will test is for which encoder layer output we will use. There are a total of 12 encoder layers, and each layer gives an output we can use to create the embedding. The results from this test are shown in figure 4.1. Both title and abstract are used for the document string. Here we can see that the AUC steadily grows the closer we get to the 12th and last layer. However the 12th layer is not actually the best. Here we can see that the output from the second to last encoder layer gives the highest AUC score. This might be because this is closer to the inference layer, and this model is trained on downstream tasks. Because of this we will test the output from both the 11th encoder layer and the 12th encoder layer.

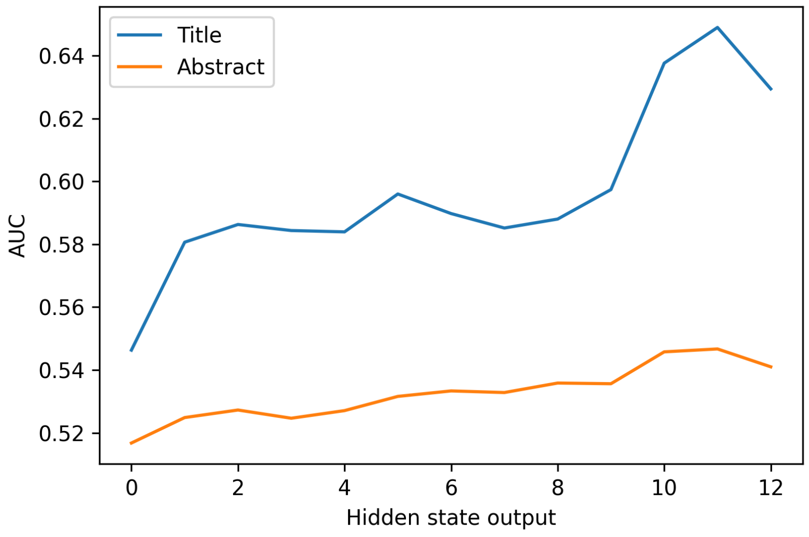

Figure 4.1: This shows the AUC for using the bi-encoder model. The x axis show which layer of the BERT model we take the embedding from.

Now we will use the cos-sim model for ranking the documents, here we will use the OR() as a matching phase. Because we can do most of the embedding in the preprocessing we can use this as a full ranking. The results from testing this are shown in the table 4.5. Just like in earlier testing we get better results from using the title than the abstract. When it comes to using the last-hidden-state VS the 11th hidden the results are not clear. The 11th layer is better for Recall(10) and MRR(10) for both title and abstract. The metric where the 12th layer output is better is for nDCG(10) using the title.

Model	Recall(10)	nDCG(10)	MRR(10)
Cos-Sim(title)	1.81E-3	**0.448**	0.426
Cos-Sim(abstract)	1.39E-3	0.380	0.379
Cos-Sim(title,hidden state=11)	**1.90E-3**	0.426	**0.493**
Cos-Sim(abstract,hidden state=11)	1.51E-3	0.401	0.435

Table 4.4: This shows the results from using the Cos-Sim as a ranking. Here we have tested the title and abstract for the document string. Here we also use the output from the 11th and the 12th encoder layer.

We can also try out some matching functions to improve the ranking by changing the ranking function. The marching functions here will be OR(), AND() and ANN(). Here we will only use the title as the document string. Remember, the matching phase affects the ideal DCG, so it might not be completely comparable. The difference between using ANN() and OR() does not seem to be very big, so there is not much to gain here. Using the AND() instead of OR() results in a big change. Using AND() as a matching phase works much better, this is likely because it can restrict more not relevant documents. Later in this project when we refer to the Cos-sim model we mean the retrieval function with AND() as the matching phase and "cord19-bert-nsp" with max-pooling from the 12th layer as the ranking function.

matching phase	Model	Match Ratio	Recall(10)	nDCG(10)	MRR(10)
OR()	Cos-Sim(title)	0.686	1.81E-3	0.448	0.4266
AND()	Cos-Sim(title)	4.52E-2	**2.50E-3**	**0.561**	**0.599**
ANN(100)	Cos-Sim(title)	6.98E-3	1.74E-3	0.440	0.424
ANN(100) + AND()	Cos-Sim(title)	5.16E-2	1.75E-3	0.442	0.430

Table 4.5: This show the results from using the Cos-Sim as a ranking and a variety of matching phases. Here we use the title as the document field.

These results need to be compared with the results from BM25. The best results from the Cos-Sim test are still worse than the worst BM25 ranking. AND()+Cos-Sim(title) gets close to OR()+BM25(title) on the nDCG(10), but it is still far worse on the other metrics. There are likely many more possibilities for improvements using the bi-encoder model, but with the current tests we are far from the BM25 models.

4.2.2 Method 2): cross-encoder

Now we will move onto the next method. This method takes in the query string and the document string at the same time. This means that this can not be done in the pre-processing, therefore we can not use this as a full ranking in a feasible amount of time. When we test the model we will use this as a re-ranker. This gives a more realistic view on how it can be used in a real application.

4.2.3 Training of BERT

The BERT model is already trained on the NSP problem. Here we can see if training improves the model. In the first test we use the $\text{BERT}_{\text{BASE}}$ model with the NSP output. The alternative model is trained on title-abstract pairs, but the structure of the model is the same. The results from this give an AUC score of 0.610 for the base model and an AUC score of 0.731 for a model trained on title-abstract pairs. The model we trained on the title-abstract pairs improves the classification. This is not so surprising considering that the base-model is not trained on the CORD-19 dataset, and the base model is only trained ont the next-sentence-prediction task.

Proportion of false examples

In the training data we can choose the probability of creating false examples p, that means that the title and abstract does not correspond. Is it good to have few or many false examples per positive? In figure 4.2 we have plotted the AUC based on the false example probability in the dataset. It is unclear what the "perfect" false example probability is. We can see that when there are too many or too few false examples it hampers the performance. Here we can see that using the abstract as the document string is better than the title.

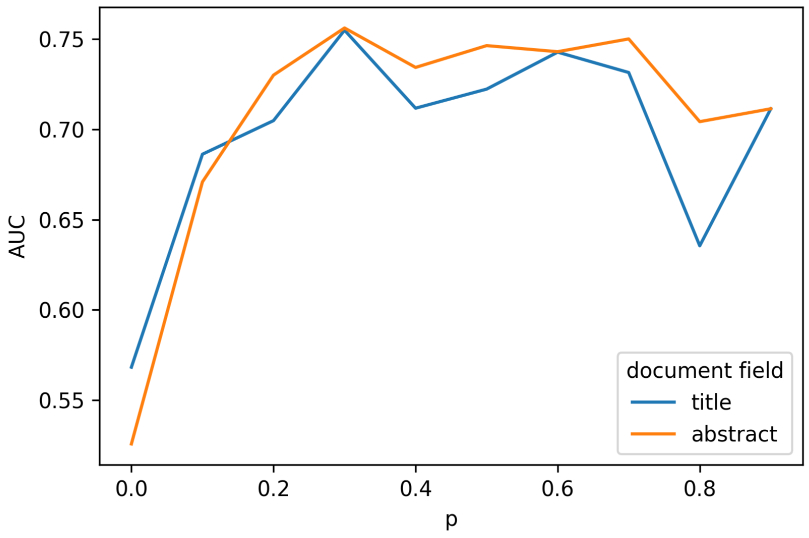

Figure 4.2: This shows the AUC for different proportions of false examples. It seems like we need enough false examples for proper training.

We can go into a bit more detail on these results. Here we are going to use a false probability rate of $p = 0.3$ and $p = 0.7$. The results from this are shown in figure 4.3. Here we take a query and document pair and the model tries to predict if it is relevant or not. This means that $f_{\text{BERT}}(q_i, d_j) = s$, then we plot the distribution of the value of s. Here we have used the true relevancy score as the color. The relevant documents are colored orange and the not relevant are colored blue. Naturally we want all the orange values close to 1, and all the blue values close to 0. We can see that this is not the case. Most values are clumped around 0 or 1. In the upper left we see that most documents are clumped around 1, even the not relevant ones. There is a slight separation, but it is not very easy to see. The best result might be in the lower right corner, here we have managed to remove many not relevant documents. There are still a lot of falsely relevant documents, so it is far from perfect.

One other issue is that the clumping seems much better when we use the abstract than the title. With the abstract we can see that there are some clear spikes around 0 and 1. There are still quite a few false positives. While it is not easy to know what the best model is from looking at figure 4.2 and figure 4.3, when ranking documents we usually care about the top results. Imagine a retrieval threshold of 0.5, then $p = 0.7$ will give a higher precision at the expense of a bit lower recall. This is a worthwhile trade-off because we care more about the precision of the top few results. Moving forward we will use a value of p in the range of [0.6,0.7].

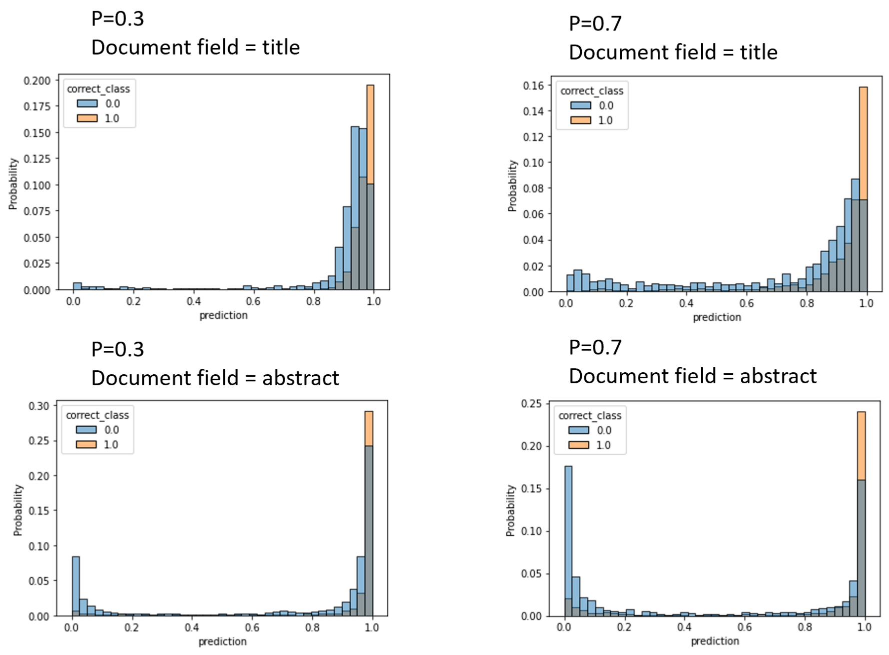

Figure 4.3: This shows the distribution of $f_{\text{BERT}}(q_i, d_j) = s$. Here the relevant and not relevant have different colors, orange is relevant and blue is not relevant. Here we use either the title or the abstract as the document field d_j. The P is the false positive rate in the training examples.

4.2.4 Perturbation test

We want to go into more detail on how to perturb the dataset for the best training. The perturbation of the dataset is described in section 3.3.2. Here we are going to train the models from the same state and on the same number of examples, but due to the randomness of the perturbation it will not be the same. The probabilities for perturbation is set as follows:

- Stop words probability: 0.4
- Short title: 0.2
- Short abstract: 0.3
- Shuffle title: 0.2

There are many ways to tweak these parameters. For this test we will use these values and either turn the perturbation on or off. The training takes around 2 hours per model. Then the model is evaluated on a subset of TREC-COVID, this means that the models are tested on the same examples. The examples consist of the query sting and the title or abstract as the document string. Here we are using the AUC to test which model is the best. The results from this are shown in the table 4.6. Here we have only included results from using the abstract as the document string, the full result is shown in the appendix A. The insights from the title and abstract seem to be the same. The results from this test are not completely clear. In general it does seem that we can improve the training using perturbation of the dataset, but it can also reduce the performance. It seems that shortening the sequences and removing stop words does help. However, shuffling the title might not work, and can actually reduce the performance. When we look at interactions of different perturbation methods the results become a bit more difficult to interpret.

There is a lot more to test in the degree of perturbation. We can change the likelihood of making a perturbed example and how severe the perturbation is. Because of the training time we can not test every conceivable combination. These results show that introducing perturbation can improve the model on the classification task, but we can also make it worse, so we have to be careful. Now the question is; can we translate the improvements in the classification task to the ranking task?

Perturbation Short sequence	Stop words	Shuffle title	Metric document field	AUC
False	Flase	False	abstract	0.7000
True	Flase	False	abstract	0.744
False	True	False	abstract	0.722
False	False	True	abstract	0.696
True	True	False	abstract	0.726
True	False	True	abstract	0.737
False	True	True	abstract	0.643
True	True	True	abstract	0.734

Table 4.6: Results from combining BM25(title) and Cos-sim(title)

4.2.5 ReRanker results

Now we have trained and evaluated the model on a classification version of TREC-COVID. What we really want is to evaluate the model on the ranking task. Here we will use this model as a second ranking phase. This works by using the results from another ranking function, like BM25w, and reordering the documents. We want the model to move the relevant documents further up and move the not relevant documents down

The results from this re-ranking are shown in the table 4.7. Here we show the first ranking function with a corresponding nDCG, the second ranking function with the query string used and a nDCG(10) score after the re-ranking. Here we have re-ranked the top-10 results. From this table the BERT re-ranker improves the results from the BM25w when using the **abstract** or **title+abstract**, however it is a bit worse when using the **title**. It also improves the results from the Cos-Sim(title) model. The earlier tests used the ranking function on tens to hundreds of thousands of documents for each query. Here we only ranked 10 documents for each of the 50 topics. This means that we have to be more careful when we test if the re-ranker improves the results. If we want to we can pick and choose different models to show a good or bad result.

Query field	First ranking model	nDCG(10)	Re-Ranker model	Document string	nDCG(10)
query	BM25w	0.658	CORD19-nsp	title+abstract	0.713
query	BM25w	0.658	CORD19-nsp	title	0.695
query	BM25w	0.658	CORD19-nsp	abstract	0.646
question	BM25w	0.714	CORD19-nsp	title+abstract	0.717
query	Cos-Sim(title)	0.558	CORD19-nsp	title+abstract	0.620

Table 4.7: Results from reranking the top-10 results using the cross-encoder.

The nDCG(10) with 10 re-ranked documents is just a snapshot of how the ranking works. Instead of just re-ranking 10 documents we can reorder X documents where $X \in \{10, ..., 100\}$, and take the nDCG(10). Here we are not too interested in the absolute value of nDCG, rather we want to know the difference between the first ranking phase and the second. Therefore, we plot $d_i = \text{nDCG}(10)_{\text{BERT}} - \text{nDCG}(10)_{\text{BM25}}$, where i is the number of documents re-ranked. The results from doing this are shown in figure 4.4. Here we have used the **title**, **abstract** and **title+abstract** as the document string. When using the **abstract** and **title+abstract** we see that even if we saw an improvement when re-ranking 10 documents this is not the full picture. While it does perform better when reranking 10 documents it decreases with more documents reranked. When using the **title** the reranker works worse. This might be because of an issue with the training, we mostly use title-abstract pairs, so the model might undervalue shorter strings.

Whether we look at the query or the question the improvements are quite small. Because we only have 50 topics a slight difference in performance can very well just happen by chance. Therefore we have added a 95 % confidence interval to the plots. The 95 % confidence intervals rearly cross the line at 0. It is not reasonable to say that the BERT model is better or worse. Because the results are so similar we might assume that the BERT model does not move the documents much, however this is not true. If we re-rank the top 10 results the documents move over 3 spots on average. This means that the BERT model has different preferences than BM25. The most robust re-ranker would probably combine BERT and BM25w. Here we could combine the strengths from BERT and BM25. If we try this on this dataset we can get a slight increase in the nDCG.

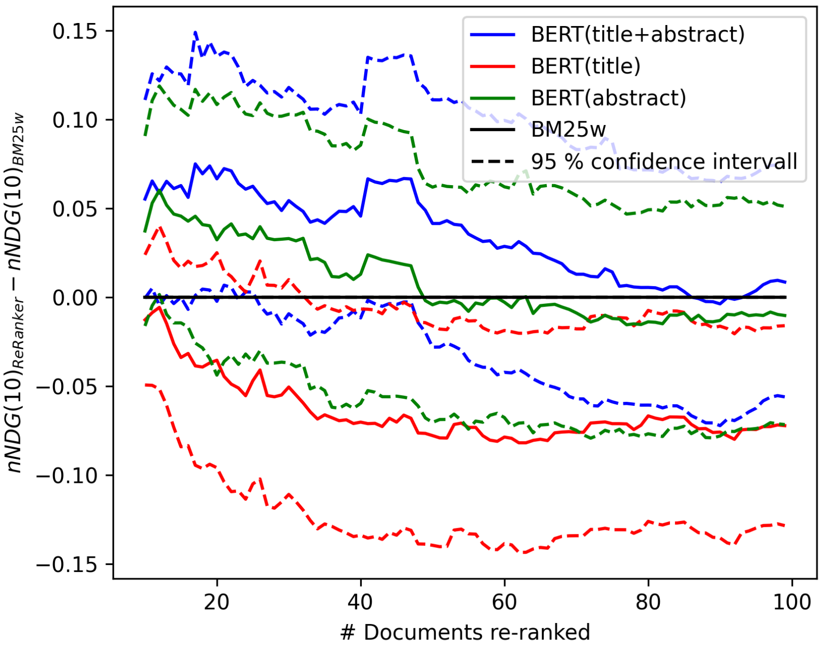

Figure 4.4: The shows the change in nDCG(10) by reranking X documents using the BERT cross-encoder model. The BERT model works slightly worse for the query, and slightly better using the questions.

To illustrate the random element a bit more we can try many different models trained with different conditions. The models have a name on this form "cord19-nsp-p=a_sw$b$_sh$c$_seq_d_e. The p is the proportion of false examples, **sw** stands for **stop words**, **sh** stands for **shuffle title**, and **seq** stands for **short sequence**. The values b, c, d and e are probabilities for doing these perturbations. The results from comparing 10 different BERT models with BM25w can be seen in figure 4.5. The **title+abstract** was used for the query field. There are some models that perform better than BM25, and some are worse. The fact that one of the models perform consistently better than BM25w can give us some hope. When trying to replicate this model with slightly different inputs we do not see the same result. For some of these models we have not used title-title pairs, or abstract-title pairs. The main point here is that small changes in the training can have a big change in the performance. If we did not have the TREC-CORD dataset available we could easily reduce the performance, because we would have no guide for what perturbation works well.

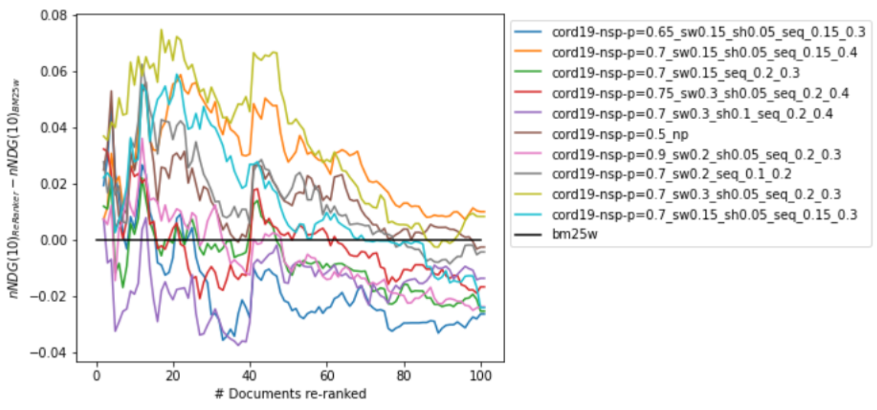

Figure 4.5: The shows the change in nDCG(10) by reranking X documents using the BERT cross-encoder model. The BERT model works slightly worse for the query, and slightly better using the questions.

Lastly we will take some time and compare the bi-encoder model and the cross encoder model. The bi-encoder used here is the model that performed best from section 4.2.1. Here we run a similar test like we did with BM25W. Here we re-rank the top k results. The results from this are shown in figure 4.6. Here we have also added a random re-ranker to illustrate how well the models perform. This shows that the cross-encoder works significantly better than the bi-encoder. And for illustration the random model works significantly worse than the bi-encoder.

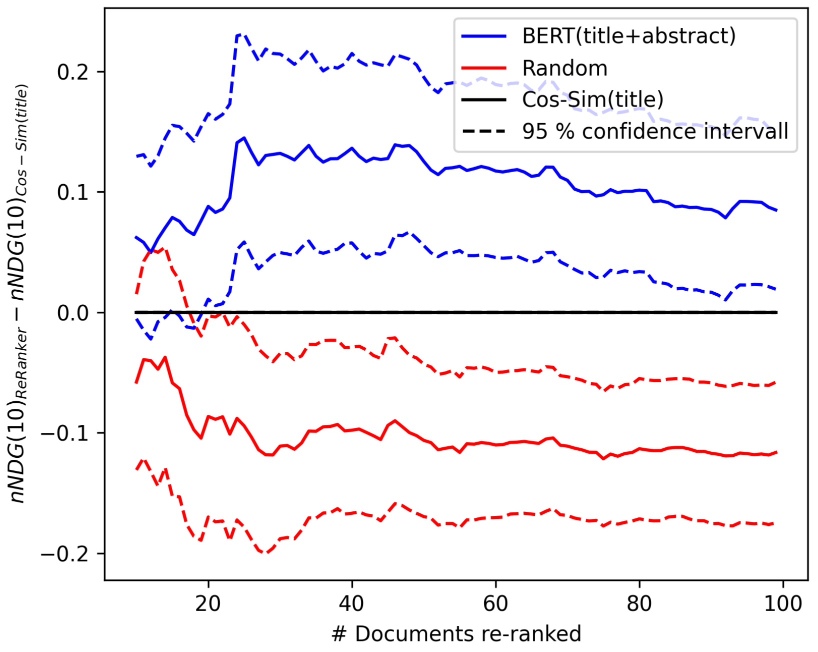

Figure 4.6: This is the result of re-ranking the top k results from the Cos-Sim model. The re-ranker model here is BERT trained on title-abstract pairs and a random ranker. The nDCG score is relative to the Cos-Sim model. Here a 95 % confidence interval is added.

4.3 Can we trust these results?

In section 2.2 we discuss the limitations of the evaluation dataset. The relevancy score for the documents is very important for evaluating the dataset. Therefore we should take a critical look at some results from the search engine. Is there a big issue with documents being marked not relevant, but really are relevant?

The first thing we can look at is if many of the top ranking documents do not have a relevancy score for that topic. Here we look at the top 100 results using BM25w. The results from this are shown in figure 4.7. Here we can see that many of the documents are assumed to be not relevant, but we do not really know if this is true. In the top 100 results over 50 % have assumed not relevant scores, in the top 10 this is reduced to 40 %. Because small changes in the top few results can make a change in the nDCG value it is very possible that these assumed 0 zeros have a great impact on the evaluation. We can not say that documents with no evaluation should be relevant, but we should look a bit more into it.

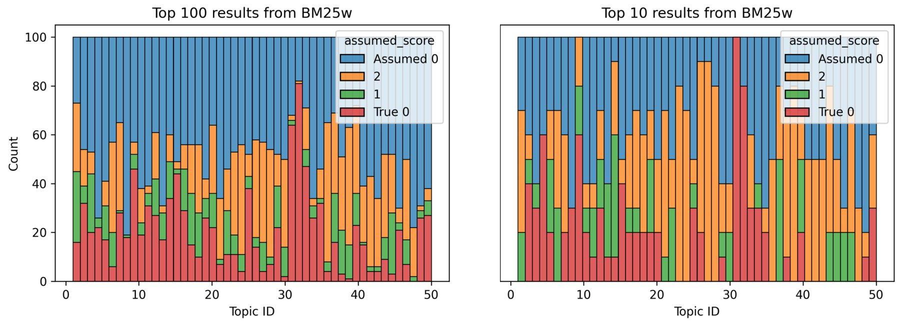

Figure 4.7: This shows the relevancy score for the top 100 results on the left, and top 10 on the right, using BM25w. The blue documents are interesting here because we do not know if they are relevant.

Now we want to see how different the results look after re-ranking the top 100 results from each topic and looking at the breakdown of the top 3 results. This is shown in tabel 4.8. This accounts for around 47 % of the nDCG(10) score. The part that can make us a bit sceptical is that the reranked results have more relevant documents, more partially relevant and fewer not relevant. Most of the not relevant are replaced with the ambiguous assumed not relevant. This is a bit worrying.

Model	# Assumed 0	# True 0	# True 1	# True 2
BM25w	54	29	16	51
bi-encoder	64	13	19	54

Table 4.8: The relevancy score for the top 3 documents different ranking models. "Assumed 0" means that we do not have a relevancy score for this document, therefor it is assumed to be zero.

Here we are going to do a test comparing BM25 results and the re-ranked results. One important assumption was that even if the TREC-COVID dataset has some limitations this will penalize the models somewhat equally. We can try to test the re-ranking but remove all documents with no relevancy score. The results from this can be seen in figure 4.8. On the left we use the same model as the one shown in figure 4.4, and on the right we use the same model as shown in figure 4.5. If we take a look at the plot to the left first. Here we see that the results are relatively better for the BERT model. Using the **title+abstract** and **abstract** as document strings we could say that the model performs significantly better than BM25w. With the **title** it also performs better, but not as much. The plot to the right shows the same trend. Here we have used the **title+abstract** as the document string. We can see that every model outperforms BM25w. This is not to say BERT is definitively better, but it does put one of our general assumptions about the dataset under question. Is BERT more affected by the assumed relevancy scores? It might seem like it. However it might also be the case that BM25w rightfully ranked the assumed documents lower. This question is very hard to answer without looking through many of the documents, and we do not have that capacity.

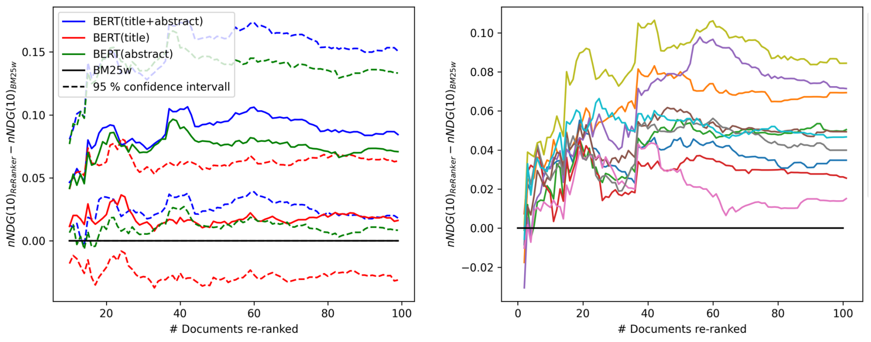

Figure 4.8: This is the results of re-ranking the top k results from BM25w. Here we compare the nDCG score of the re-ranker model with the BM25w model. On the left we compare using different document strings, and a 95 % confidence interval is added. In the right plot we test many different models with different training parameters. For both test every document with relevancy ”Assumed 0” is removed.

If we take all results into consideration it is not clear if BERT improves the BM25w ranking. The first result with ”assumed 0” included showed some improvements, but this was withing the 95 % confidence interval. When we removed the ”Assumed 0” the BERT model was significantly better, but we can not verify if this is fair. These two results leads us to thinking that the cross-encoder improves the ranking of BM25, at least when using both title and abstract.

The other issue is wetter this training is consistent and reproducible for different problems, especially problems with no evaluation dataset. We uses the TREC-CORD to guide the choice of training dataset hyper-parameters. Small changes could be the difference between a model than improves the ranking and a model that makes it worse. It might be difficult to reproduce this result without a validation dataset. A BERT cross-encoder model trained on title-abstract pairs *can* become better than BM25, but with small changes to the training dataset it can also preform worse than BM25.

Chapter 5

Conclusion

Using the BERT bi-encoder model for the ranking model showed several methods for improving the ranking with this model. It was clear that training the model on the CORD-19 dataset improved the ranking. The choice of the pooling method also affects the performance. On the classification task the best max-pooling model achieved an AUC= 0.67, the best cls-polling got an AUC= 0.62 and the best mean-pooling got AUC= 0.59. We also found that what layer of the BERT model we took the embedding from changed the outcome. The 11th layer worked better than the 12th layer on the classification task. However on the ranking task it was not as clear. Here the 11th layer worked better on the MRR(10) and Recall(10), but the 12th layer worked better on the nDCG(10). If we compare the bi-encoder with the BM25 model, then we see that the bi-encoder worked worse than the BM25 models.

Using the cross-encoder model trained on title-abstract pairs gave many different results. We used the classification problem to guide how to perform the perturbation. Here we found models trained on the title-abstract pairs worked better than the base models with a pre-trained NSP-layer. Different perturbations to the title-abstract pairs improved the performance on the classification task. Randomly shortening down the title and the abstract worked the best. When applied as a re-ranker to the bi-encoder model we found that the cross-encoder worked significantly better. When reranking the BM25w results it became a bit more difficult to interpret. When the cross-encoder reordered the top-10 results from BM25w it got a better nDCG score. This result was within the 95 % confidence interval, and the performance reduced the more documents were evaluated. In an application it is hard to say that adding this BERT model as a re-ranker is worth it, mainly because of the additional computation required.

In the data section we assumed that models would be somewhat equally affected by including documents with no relevancy score. This might not be a good assumption, it seems that the BERT model where more affected by the assumed scores. It is hard to say this conclusively for two reasons; one with only 50 topics we have to account for some randomness and two it is hard to evaluate every document with no relevancy score. Because of these issues it is hard to separate models with similar performance, whether the model is slightly worse or slightly better. The conclusion we can come with is that using BERT for reordering the top results from BM25w gives similar performance.

Further research

There are parts of this project where we can research more or expand upon. Here we will go through some ideas that might improve the performance of the search engine. This performance could be in terms of better ranking or reducing latency.

Firstly we could try to use BERT to improve BM25. One of the issues with BM25 is that it does not work well with synonyms. One idea could be using BERT to generate semantically similar sentences, then we can use these semantically similar searches in the BM25. The BERT model generates n new queries Q_i from the original query Q. Then we get the BM25* score will become

$$\text{BM25}^\star(Q, D) = BM25(Q, D) + \sum_{i}^{n} w_i \cdot BM25(Q_i, D) \tag{5.1}$$

We have added a weight, this could be how semantically similar the alternate query is to the original query. Naturally this is going to be more computationally expensive, and the model will have to justify this by improving performance.

The next part we could make improvements for is the Cos-Sim model. Remember, these models were not directly trained on the retrieval task. Training the BERT embedder specifically for information retrieval. We could also use a more complicated ad-hoc model than the Cos-Sim.

A big part of this project was focused on generating training data for the information retrieval. It seems that generating examples this way is difficult, and different perturbations can have a big effect on the performance. Having more realistic examples could make the training easier and more consistent. There are many other good datasets for training models on information retrieval, one example is MS MARCO. This dataset contains realistic query-document pairs. It is possible to train the model on the MS MARCO dataset, then only do a little fine tuning on the TREC-COVID dataset. This could combine a little bit of both.

Another idea is to modify the loss function used. When training the BERT model we used a point wise loss function. That means that we took one query and one document and computed the loss on one instance at the time. There are several other possibilities for loss functions. One can be a pairwise loss function. That means that the goal is to determine if document A is more relevant than document B, or is it the other way around. One can take it a step further and make a listewise loss function. In a paper training deep learning models on MS MARCO (Han et al. [2020]) listwise loss functions worked better than pairwise, which again worked better than the pointwise loss function.

There has been a lot of talk about how complex the BERT model is, and how slow it is as a consequence. While deep-neural network models will be more complicated than models like BM25 it is possible to reduce the latency. This can be done by distilling the BERT models. There are many who have done this. One example uses a model with 12 layers and uses it as a teacher for a model with 6 layers (Sun et al. [2019]), this means cutting the number parameters in half. The performance only drops by 1 - 2 % on 5 different language tasks. This shows that it is possible to reduce the complexity of BERT models, while still keeping much of the performance. Because search engines have to rank the documents when a query arrives this reduction in performance is probably worth the increased speed. If the BERT model does work better than BM25 then being able to rank more documents in the same amount of time, therefore increasing the performance.

Bibliography

Vespa python api¶. URL https://pyvespa.readthedocs.io/en/latest/index.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Derek Eder. URL https://microsoft.github.io/MSMARCO-Passage-Ranking-Submissions/leaderboard/.

Luyu Gao, Zhuyun Dai, and Jamie Callan. COIL: revisit exact lexical match in information retrieval with contextualized inverted list. *CoRR*, abs/2104.07186, 2021. URL https://arxiv.org/abs/2104.07186.

Arash Habibi Lashkari, Fereshteh Mahdavi, and Vahid Ghomi. A boolean model in information retrieval for search engines (pdf). *Information Management and Engineering, International Conference on*, 0:385–389, 01 2009. doi: 10.1109/ICIME.2009.101.

Shuguang Han, Xuanhui Wang, Mike Bendersky, and Marc Najork. Learning-to-rank with BERT in tf-ranking. *CoRR*, abs/2004.08476, 2020. URL https://arxiv.org/abs/2004.08476.

Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. Pretrained transformers for text ranking: Bert and beyond, 2020. URL https://arxiv.org/pdf/2010.06467.pdf.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Yury A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016. URL http://arxiv.org/abs/1603.09320.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R Hersh. TREC-COVID: rationale and structure of an information retrieval shared task for COVID-19. *Journal of the American Medical Informatics Association*, 27(9):1431–1436, 07 2020. ISSN 1527-974X. doi: 10.1093/jamia/ocaa091. URL https://doi.org/10.1093/jamia/ocaa091.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. *CoRR*, abs/1908.09355, 2019. URL http://arxiv.org/abs/1908.09355.

Vespa team. Approximate nearest neighbor search using hnsw index edit page. URL `https://docs.vespa.ai/documentation/approximate-nn-hnsw.html`.

team Vespa. Quick start edit page. URL `https://docs.vespa.ai/documentation/vespa-quick-start.html`.

trec covid, 2020. URL `https://castorini.github.io/TREC-COVID/round5/`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018. URL `http://arxiv.org/abs/1804.07461`.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, K. Funk, Rodney Michael Kinney, Ziyang Liu, W. Merrill, P. Mooney, D. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, B. Stilson, A. Wade, K. Wang, Christopher Wilhelm, Boya Xie, D. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. Cord-19: The covid-19 open research dataset. *ArXiv*, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL `http://arxiv.org/abs/1910.03771`.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015. URL `http://arxiv.org/abs/1506.06724`.

Appendix

Appendix A

Appendix

A.1 NLP basics

A Corpus

A corpus is a collection of documents or strings. For this application the corpus is the total set of documents in the CORD-19 dataset.

B Tokenizer

Text strings has to be broken into discrete tokens. Usually these tokens represent words or symbols. This is often a useful way to segment strings. It is usually easier for models to utilize tokenized strings rater tan take in strings character for character.

C Stemming

Stemming tries to return words to the root form. This is done by removing endings of words, "running" becomes "run". The motivation to do this is reducing complexity and make the strings more uniform.

This can be very usefully for search engines. The uses does not always use the same form of a word as the document. For example, the user can search for "How to fix a bike", and the relevant document can contain the string "Fixing bike tutorial". While "fixing" and "fix" is not the same word, but they will have the same stemmed form. For a term matching algorithm it is easier to match the stemmed words.

Removing endings is still loss of information. In models like BERT we want to save these endings and utilize it.

D Stop-words

Stop-words is usually the most frequent words in a language. These are words like "a", "but", "and" and so on. In some application we remove these stop-words, and this can be done without a loss in performance. The list of stop-words can range from tens to thousands.

The issue is that stop-words often carry some semantic meaning. With increased computing power it is not really as necessary to remove stop words. Removing large lists of words has gone out of favour.

E OOB - Out of Box

Using a piece of software out of the box. In this case it means that we do not train the model before applying it to this problem.

A.2 Common abbreviations

- KNN: k nearest neighbours
- ANN - approximate nearest neighbours
- BERT - Bidirectional Encoder Representations from Transformers
- TF-IDF - term frequency-inverse document frequency
- ROC - receiver operating characteristic curve
- AUC - area under cure
- NSP - next sentence prediction
- MLM - masked language modeling

A.3 Stop words

In the following weird paragraph we list all the stop words used in this project:

- a about after all also always am an and any are at be been being but by came can cant come could did didn't do does doesn't doing don't else for from get give goes going had happen has have having how i if ill i'm in into is isn't it its i've just keep let like made make many may me mean more most much no not now of only or our really say see some something take tell than that the their them then they thing this to try up us use used uses very want was way we what when where which who why will with without wont you your youre , ! $ % ' ( ) * + , - . / : ; ¡ = ¿ ? @ [ ]

A Perturbation test

Full results of perturbation test as described in section 4.2.4. Here we have also added the results from the title.

B Re-Ranking

Here we have added some additional results from the re-ranking tests. In section 4.2 we added some plots where we compared BERT as a re-ranker with BM25w. It was not clear what model was better. Here we add some more tests. This is in part to show that we can get some clear results, The first test shown in figure A.1 we have compared BM25w with "bert-base-uncased", that is a model only pre-trained on general NSP tasks. The BERT model starts out quite similar to BM25w, but the more documents it ranks the worse it gets. Here we can say with some confidence that BM25w works better than "bert-base-uncased".

The next test we compare the BERT model trained on title-abstract pairs with the Cos-Sim(title) model. We have also added a random ranker, this models shuffles the top k documents around. The results from this is shown in figure A.2. Here we can see that the BERT-NSP model works

Perturbation Short sequence	Stop words	Shuffle title	Metric document field	AUC
False	Flase	False	title	0.647
False	Flase	False	abstract	0.7000
True	Flase	False	title	0.6962
True	Flase	False	abstract	0.7437
False	True	False	title	0.6862
False	True	False	abstract	0.7218
False	False	True	title	0.628
False	False	True	abstract	0.6957
True	True	False	title	0.67858
True	True	False	abstract	0.7258
True	False	True	title	0.7083
True	False	True	abstract	0.7367
False	True	True	title	0.6744
False	True	True	abstract	0.6434
True	True	True	abstract	0.7335

Table A.1: Results from perturbation the perturbation test

better than the Cos-Sim(title). We can also see that the Cos-Sim(title) works better than the random re-ranker.

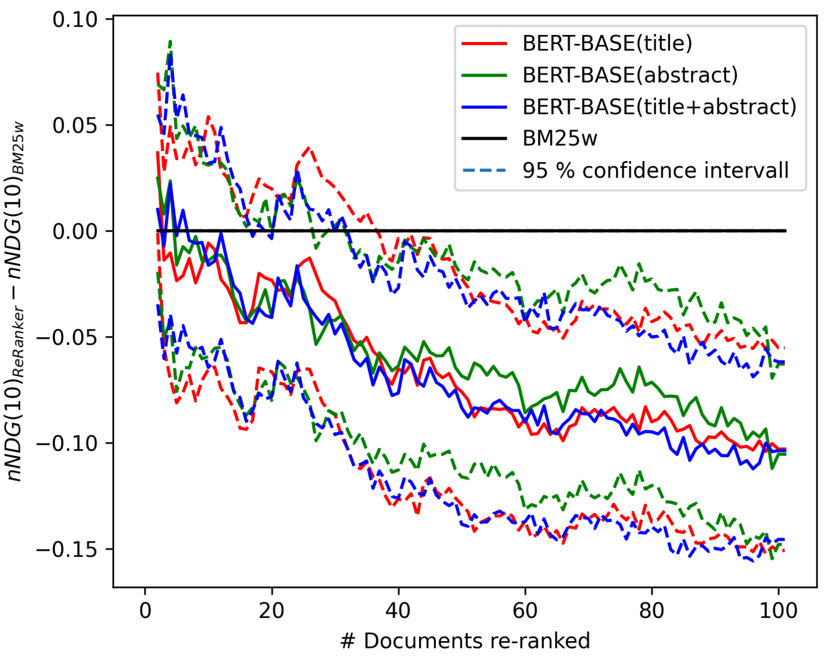

Figure A.1: This is the results of re-ranking the top k results from BM25w. The re-ranker model here is **bert-base** with a NSP inference layer. Here we compare the nDCG score of the re-ranker model with the BM25w model. Here a 95 % confidence interval is added.

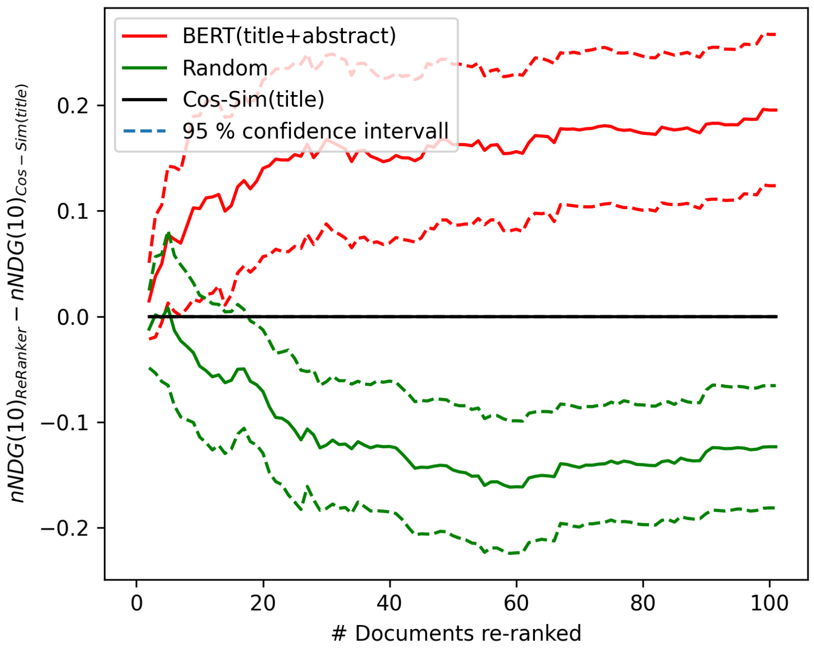

Figure A.2: This is the results of re-ranking the top k results from the COS-SIM model. The re-ranker model here is BERT trained on title-abstract pairs. The nDCG score of the re-ranker model with the BM25w model. Here a 95 % confidence interval is added.