University of Bergen
Department of Informatics

Document Ranking for Systematic Reviews in Medicine

Author: Simen Høyvik
Supervisors: Nello Blaser, Øystein Ariansen Haaland

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

May, 2023

Abstract

This master thesis delves into the exploration of document ranking models, with a particular focus on architectures based on BERT (Bidirectional Encoder Representations from Transformers). The primary objective of the thesis is to train these ranking models using textual data extracted from systematic reviews. The purpose is to develop a tool that can assist researchers in conducting systematic reviews by prioritizing the most relevant studies, thus eliminating the need to screen non-relevant studies when looking for evidential information. The study includes the creation of a labeled dataset from a collection of systematic reviews by fetching additional textual content from PubMed. In addition, the study presents a training framework for training various document ranking models. The results indicate that the models can capture the underlying patterns within the training data. However, they exhibit suboptimal performance when presented with unseen data. This overfitting phenomenon could be attributed to the dataset's dissimilar writing styles. Furthermore, the similarity between the text in relevant and non-relevant studies might also contribute to this performance disparity.

Acknowledgements

I want to thank my supervisor, Nello Blaser, for his exceptional guidance and collaboration throughout this research endeavor. His expertise and support have been significant in shaping this thesis's theoretical aspects and making crucial decisions. I would also like to thank Øystein Ariansen Haaland for providing great knowledge on health-related topics. I sincerely thank the Cochrane Library for graciously providing access to valuable Systematic Review data. This invaluable resource has significantly enriched the depth and breadth of my research, allowing for a comprehensive analysis of the subject matter. I would also like to thank the University of Bergen for providing computer resource facilities. The availability of these facilities has played an essential role in enabling the training of larger-scale language models. I would also like to acknowledge and thank my fellow co-students for their camaraderie and support. The journey through the master's program has been enriching, largely due to their insightful discussions, shared experiences, and friendships. Lastly, I am grateful for my family and friends' constant support.

Simen Høyvik
Tuesday 30th May, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The World Health Organization (WHO) is a worldwide organization trying to promote health, keep the world safe, and serve vulnerable populations [25]. Despite its efforts, a significant portion of the global population lacks access to essential health services, especially in low and middle-income countries. This has a significantly negative impact, leading to an alarming number of deaths or unnecessary loss of life. As for now, at least half of the people in the world do not receive the health services they need, and about 100 million people are forced into extreme poverty, mainly due to spending out-of-pocket money on health care [26].

The United Nations (UN) are promoting better living standards and human rights for everyone. They have therefore developed the Sustainable Development Goals (SDG) to transform our world. SDG Target 3.8 relates to the lacking health services and extreme poverty around the world [24].

> *SDG Target 3.8: Achieve universal health coverage, including financial risk protection, access to quality essential health-care services, and safe, effective, quality, and affordable essential medicines and vaccines for all.*

A step towards SDG Target 3.8 is to offer Universal Health Coverage, meaning everyone can access the health services they need without suffering financial hardship [26]. In low-income and low-middle-income countries with limited health budgets, challenging choices must be made regarding including medical interventions in health benefits packages (HBP), encompassing the government's provision of specific interventions, either free or at a reduced cost. Due to some countries' limited money and resources,

deciding which intervention to include in the HBP is difficult. In determining the inclusion of interventions in a health benefits package (HBP), cost-effectiveness—calculated as the ratio of cost to effect—is typically the primary criterion. Consequently, having accurate and comprehensive information regarding all interventions' costs and health effects is crucial for transparency and fairness. Insufficient information increases the risk of excluding cost-effective interventions from the HBP or including interventions that are not cost-effective. If the government should fulfill SDG 3.8, it is essential to prioritize interventions transparently and fairly.

An HBP can be developed by employing strategic planning and management tools [23, p. 10]. One such management tool is the University of Bergen's FairChoices - DCP Analytics Tool, from now on referred to as FairChoices. FairChoices is a user-friendly tool intended to assist countries with limited resources in prioritizing investments in the health sector. Its main function is to estimate different parameters of interests, such as cost-effectiveness, equity impact, and financial risk protection of various medical interventions [22]. Medical intervention is any action to enhance human health, whether it involves preventing illnesses, treating or mitigating the severity or length of an ongoing disease, or reinstating function compromised due to an injury or illness [33, Ch. 2]. For FairChoices to do this, it is entirely dependent on key input parameters for a particular medical intervention. Key input parameters are usually found in health study publications and summarized in Evidence Briefs.

Evidence Briefs are used as input data for the FairChoices tool to estimate parameters of interest. For FairChoices to evaluate every existing intervention, it needs input data from every existing intervention, meaning it needs an Evidence Brief from each medical intervention. The main information included in Evidence Brief concerns the cost and effect of the particular intervention. Costs are measured per intervention for one person for a whole year. Effect represents improved survival if you get infected by the disease, improved life quality, and reduced disease occurrence. This information is retrieved from health studies literature, such as Randomized Controlled Trials (RCT), observational studies, and patient-reported outlines, which can be found in medical databases. I will, throughout the report, refer to them as documents. Currently, researchers from the University of Bergen create Evidence Briefs by conducting a Rapid Review, which involves surveying existing medical publications about a particular medical intervention. Instead of surveying all publications, the researchers survey until they have found sufficient evidence to state parameters of interest. For many interventions, an abundance of information is available. Medical literature containing new evidence is constantly being

published, and some of these publications may contain relevant information, whereas others may not. Due to the lack of existing Evidence Briefs and the fact that Rapid Reviews have many similarities to Systematic Reviews, I will mainly focus on Systematic Reviews in this thesis.

The Systematic Review approach follows a step-by-step method that summarizes evidence from a series of related documents on a particular topic. It is done in a rigorous, transparent, and standardized methodology to identify, critically appraise, and synthesize all relevant documents on a specific topic [17].

The Cochrane Library is a database collection containing different types of high-quality and independent evidence-based documents. These documents are intended to inform healthcare decision-making [15] and are used by medical researchers when conducting a Systematic Review for a particular intervention. A Systematic Review mainly includes four steps [17]:

1. Identification of relevant documents from several sources.
2. Selection of documents based on clear, predefined criteria.
3. Systematic collection of data.
4. Appropriate synthesis of data.

When identifying relevant documents, also called screening, researchers may have to evaluate thousands of articles, making the task extremely time-consuming. The average time for conducting a Systematic Review is 15 months, risking the review being outdated before it is finished [6, p. 1]. In addition, it should be noted that the proportion of relevant documents obtained during the screening phase can be as low as 1% of the overall search yield. [6, p. 1]. Search yield refers to the number of documents identified through the search process that meet the criteria for inclusion in the review. And with the ever-increasing amount of articles published, the screening stage will become an even more significant bottleneck if no tools are provided.

As mentioned, conducting a Systematic Review is complex and challenging without tools. Susan Sykes highlights a series of challenges researchers will face when starting a Systematic Review process, including logistics and coordination, human error, lack of time, and searching for solutions [34]. In this work, I have focused on the third challenge, concerning the time aspect of doing a Systematic Review of searching and

screening medical documents. As for now, the screening step is often a manual job where researchers have to browse through results from scholars. It is also known that researchers have lacked comprehensive guidance on which search systems are suitable for systematic searches [19, p. 2]. The scholar can be search-based, such as PubMed, Cochrane, or Google Scholar, where the researcher inputs a search query to a search engine, and a ranked list of documents is returned. However, these search engines are not optimized for conducting Systematic Reviews. Documents containing information needed by the researcher conducting the Systematic Review may be ranked far down the list. This is because search engines tend not to rank newly published articles as high as older papers with more citations and click-rate. This makes it more difficult for researchers to find newly published evidence relevant to the type of interventions examined in the Systematic Review. Another way to find relevant documents is to use search engines with word-matching techniques and logarithmic operators. The user then has an initial idea of which words should be included and which should not be for a study relevant to the examined intervention. The search engine will only return documents that satisfy the user's logarithmic search input. Synonyms and related terms are used to increase the sensitivity of the search. This method is also time-consuming and lacks performance due to difficulties defining appropriate inclusion criteria.

Thus, a model optimized for ranking the most related documents that include evidence to estimate parameters of interest higher than non-related documents for a particular medical intervention is needed. The primary purpose of the ranking model is to support researchers in the search and screening step of a Systematic Review.

Different document ranking models have been developed and have shown promising results. However, there are no publications on applying these document ranking algorithms to Systematic Review datasets. Why has this not been attempted? There are many reasons why document ranking algorithms haven't been trained on scientific health data. Firstly, research on document ranking algorithms has increased exponentially and has only recently proven to have great results. Thus, there has not been much time to research the possibility of training them on a Systematic Review task. Secondly, the outcome may not reflect the effort in developing the document ranking algorithm for their task. Other factors also determine why document ranking has not emerged as widely in healthcare as in other fields. Some factors include complex and diverse data, lack of standardization, limited resources, and the need for human interpretation.

1.1 Objectives

> *How can document ranking algorithms be trained to assist researchers when conducting Systematic Reviews? Is it possible to develop a useful document ranking model that assists researchers in conducting a systematic review?*

This thesis seeks to determine if existing machine learning models can be trained to assist users when conducting Systematic Reviews. The desired outcome is to rank documents based on textual input. The ranked result should provide documents that fulfill the inclusion criteria and contain evidence related to the user input at a higher rank than those that do not. Specifically, the objectives of this thesis are:

(A) To review the current state-of-the-art transformer-based BERT models and how they can be used for document ranking.

(B) To create a dataset that facilitates training neural ranking models.

(C) To design, implement, and evaluate different ways of using the BERT model, including interaction and representation-based approaches for ranking health documents, comparing their performance to other approaches such as TF-IDF in combination with Logistic Regression, BM25, and PubMed ranking.

(D) To investigate the use of various textual features in documents, including the related documents' titles and abstracts. Also, to evaluate the impact on various hyperparameters relevant to BERT-based models of the task-specific neural network.

(E) To analyze the study results, draw conclusions, and make recommendations for future research, especially in using transformer-based models, such as BERT, in ranking health documents.

By accomplishing these objectives, the thesis aims to provide an in-depth understanding of different ways of using the BERT model in the ranking of health documents and its comparison to other models. The results of this research will contribute to developing more effective and efficient tools for healthcare professionals and researchers to find relevant health documents when conducting a Systematic Review of a particular medical intervention. It will determine the possibility of using Systematic Review data from the Cochrane Library to train document ranking models.

1.2 Thesis Outline

This thesis is structured into seven chapters, each dedicated to exploring different aspects of the research question: *How can machine learning algorithms improve document ranking when conducting Systematic Reviews? Is it possible to develop a useful document ranking model that assists researchers in conducting a systematic review?*

Chapter 1 briefly introduces the topic and presents the problem: document ranking in Systematic Reviews. It outlines the research questions and the objective of the study.

Chapter 2 details all the theoretical frameworks and concepts relevant to the study. This chapter examines previous research on document ranking and sets the stage for the following empirical analysis.

Chapter 3 details the Systematic Review data used in the study, gathered from Cochrane Library and PubMed. This chapter discusses the research design and how to create a labeled dataset suitable for the task.

Chapter 4 talks about the proposed solutions for training document ranking models. It goes in-depth on implementation details and considerations. This chapter discusses the algorithms used and how they were implemented.

Chapter 5 presents the results of the study. This chapter discusses the empirical analysis, with a focus on the effectiveness of the algorithms in improving document ranking in Systematic Reviews.

Chapter 6 provides a discussion of the results and their implications. It examines the strengths and limitations of the study, as well as proposes future research directions.

Chapter 7 concludes the thesis by summarizing the key findings, discussing their significance, and offering recommendations for future research. It draws together the key themes from the study and highlights their importance in document ranking in Systematic Reviews.

Chapter 2

Background

The background chapter includes all the theoretical and technical concepts needed to understand this thesis fully. I am describing the theoretical aspects of machine learning and natural language processing. I am finishing this chapter by elaborating on related works, including neural ranking models and existing ML tools to help the systematic review process.

2.1 Machine Learning

This section discusses machine learning (ML) concepts used in this research. Many concepts are described based on the book *The Hundred-Page ML Book* by Andriy Burkov [7].

ML allows computer systems to learn and improve from experience without being explicitly programmed. It involves using algorithms and statistical models that enable a computer to learn from data and make predictions on new data. We divide ML into supervised, semi-supervised, unsupervised, and reinforcement subcategories. We will, in this section, only explain supervised and unsupervised.

2.1.1 Supervised Learning

In supervised learning, we have a labeled dataset containing information on what we want to predict in the future. We refer to this dataset as $\{(x_i, y_i)\}_{i=1}^{N}$ where x_i is the observation represented as a feature vector, y_i is the actual value (referred to as label), i is the index of the particular data point and N is the total number of data points. The feature vector x_i typically contains numerical values that describe the observed entity or phenomenon. For instance, in a classification problem of classifying customer reviews as positive or negative, the feature vector could represent the length of the review, the presence of specific keywords, the number of exclamation marks, and the average word length. The label y_i represents the class membership we want to predict, where $y_i = 1$ corresponds to a positive review and $y_i = 0$ corresponds to a negative review. If the problem involves predicting the sentiment of a review as either positive or negative, it is called a binary classification problem. In this case, the label y_i corresponds to the two possible classes: positive or negative sentiment.

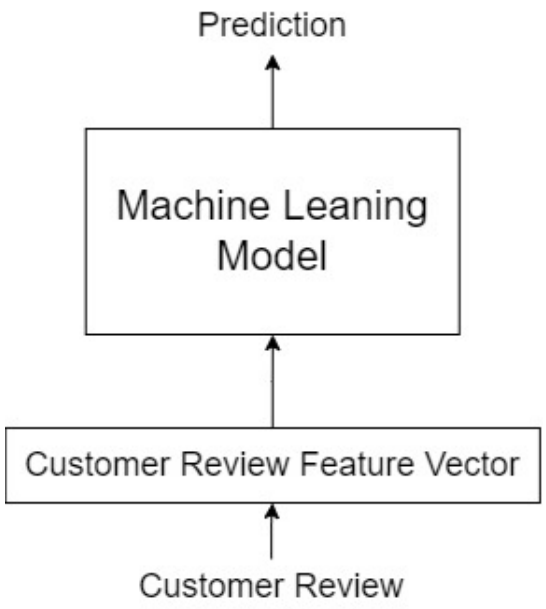

Figure 2.1: Basic architecture for customer review classification using machine learning. The customer review is inputted, and a feature vector is generated. This feature vector is then used as input for the machine learning model, which predicts the sentiment of the customer review.

Figure 2.1 illustrates how a particular feature vector x_i representing a customer review is provided as input to the classification model. The model then generates a prediction, denoted as $\hat{y}_i$, indicating whether the review is positive or negative. The prediction may

or may not match the actual label, y_i, which represents the true sentiment of the review. Supervised ML aims to train the algorithm using a labeled dataset, enabling it to learn the underlying patterns between the input feature vector (review characteristics) and the output label (review sentiment). The trained model can make accurate predictions on new, unseen customer reviews.

2.1.2 Unsupervised Learning

In contrast to supervised learning, the labels are left out for unsupervised learning. The dataset is thus a collection of $\{(x_i)\}_{i=1}^{N}$ where x_i is a single vector of data features, i is the index of a particular data point and N is the total number of data points. The primary goal of unsupervised learning is to discover patterns or structures in the data without prior knowledge of some output labels. A standard unsupervised learning method is clustering, which aims to group similar data points into clusters based on their feature vectors. Once the clustering algorithm is trained on the dataset, we can use the resulting clusters to predict the categorical class of new data points. Another type of unsupervised learning is training language models using extensive collections of raw texts. Section 2.4.6 explains these unsupervised learning methods in detail.

2.1.3 Parameters vs Hyperparameters

Standard for almost all ML models is that they have both parameters and hyperparameters. These terms are used throughout the whole thesis and cannot be interchangeable. Parameters are internal variables of the model that are learned during the training process. They represent the model's internal data representation and capture the patterns and relationships within the training dataset. If the model is trained iteratively, the values of parameters are updated through optimization algorithms. The training aims to find the optimal values for parameters that best fit the training data and generalize well to unseen data. When considering neural networks, we some times refer to the parameters as weights and biases.

On the other hand, hyperparameters are external variables set before the learning process begins. They define the structure and configuration of the learning algorithm. Examples of hyperparameters include the learning rate, dropout value, number of layers or units in a neural network, batch size, and activation functions. Hyperparameters influence

how the model learns and generalizes from the training data but are not learned from it. Instead, they are chosen by the practitioner based on intuition, domain knowledge, or through experimentation and tuning. Selecting appropriate hyperparameter values is crucial, as they directly impact the model's capacity to learn, convergence speed, generalization ability, and overall performance.

2.1.4 Machine Learning Pipeline

The following subsection comprehends the process involved in the execution of an ML project. The first step is data collection. Occasionally, you might receive a dataset from someone else or need to collect the data yourself. The dataset is used to train, select and evaluate the model. The next is data preparation and feature engineering. This step prepares the data for training by cleaning, transforming, and sometimes normalizing the data. This step is critical to ensure the data is suitable for the model and avoid errors or biases. The next step is model type selection. A model type is selected for training based on the specific problem and the nature of the data. Once a model type is selected, we can start training an instance of the selected model type. One option is to train one model with fixed hyperparameters. Unfortunately, the hyperparameters resulting in the best model are not something we know beforehand. And it is thus common to train a series of models with different hyperparameters to pick the best model. Various factors influence the best model, usually related to the problem we are trying to solve. The most common approach is to hold out a portion of the dataset before training, making it possible to evaluate how the model performs on data it has not seen before. We usually refer to this portion of the data as validation data, which is usually around 15% of the total dataset but can vary depending on how much data we have. When we have chosen a model based on some performance criteria, we evaluate the model on another hold-out portion of the dataset. We refer to this portion of the dataset as the test set, which is usually the same size as the validation set. This evaluation is needed to get an unbiased estimate of how the chosen model performs. Section 2.1.8 describes different performance measures, especially related to document ranking.

2.1.5 Generalization

Generalization is a critical aspect when training ML models, which refers to the ability of the model to perform well on new, unseen data. The goal of training a model is not

to memorize the training data but to learn patterns that can be applied to new data. Suppose you want to build an ML model that can classify movie reviews as positive or negative based on their text. Consider that the dataset is in the same format as explained in 2.1.1. Now, you train a neural network on this dataset to achieve high accuracy on the training data. The model learns to recognize patterns in the text, such as the words' sentiment, the review's tone, and the sentences' length. However, if the model memorizes the training data, it cannot generalize to new, unseen reviews. For example, suppose the model learns to recognize a specific set of words or phrases overrepresented in the training data. In that case, it may perform poorly on reviews that use different word phrases to express the same meaning. The model must learn to recognize more general patterns found in training and validation data to achieve good generalization. For instance, the model should be able to identify sentiment-carrying words, such as *good*, *bad*, *great*, *terribly*, and so on, and understand how they contribute to the overall sentiment of the review. However, the model should also consider that if the word *terribly* is used in combination with *good*, this is a positive indication. In summary, the model must learn general patterns that apply to many scenarios, not just the specific examples in the training data to achieve generalization.

2.1.6 Overfitting vs Underfitting

The bias-variance tradeoff is significant in ML. We say that when the model performs poorly on the training and validation set the model is underfitting. The model is thus not able to learn anything from the training data. This behavior can occur if the model is too simple or the engineered data features are not informative enough. Contrarily, overfitting happens when the model performs well on the training data but poorly on either validation or test data. The model has then learned the patterns in the training data too well and cannot generalize to unseen data. This behavior can happen if the model is too complex or the data contains too many features and insufficient data samples. It's about finding the right balance between two types of errors: underfitting and overfitting. To achieve good generalization, a balance between simplicity and complexity is needed. Various models with different hyperparameters must often be trained to find the best.

2.1.7 Logistic Regression

Logistic regression is a supervised learning model used for binary classification tasks. The goal is to predict a binary output (e.g., yes or no, 1 or 0) based on input features. In

logistic regression, the output is modeled as a function of the input features, and the goal is to learn the parameters of this function that best fit the data.

Let's define how logistic regression works using a single data point. We have from section 2.1.1 defined a single data point for supervised learning as a x_i, y_i pair where in this case x_i is a numerical vector and y_i is either 0 or 1. The logistic regression model assumes that the probability of the positive class (e.g., yes, 1) is a logistic function of a linear combination of the input features using a logistic function with the name sigmoid:

$$\hat{y} = P(\hat{y} = 1|z) = \frac{1}{1 + exp(-z)} = simgmoid(z) \tag{2.1}$$

where $\hat{y}$ is the binary predicted output and z is a linear combination of the input features weighted by a set of learned parameters. This function will output a single numerical number representing the probability of x_i being true. We can then say that if the value is above 0.5, the final prediction is 1. If the value is below, then the prediction is 0. Lets look closer at how z is calculated:

$$z(x_i, w) = \sum_{j}^{K} x_{ij} * w_j \tag{2.2}$$

where x_{ij} is a specific value in the feature vector, w is the learnable weights defined as a vector, and w_j is a specific learnable weight. K is the total number of specific features in x_i. The sigmoid function is applied to this z value to obtain the probability of the positive class. You can see that the sigmoid function maps any real-valued number to a value between 0 and 1, which makes it helpful in modeling probabilities or binary classification tasks.

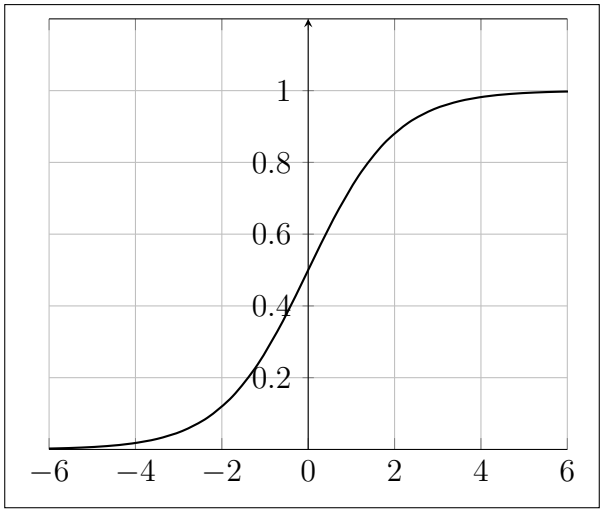

Figure 2.2: Graph showing how sigmoid function maps any positive or negative number to a number between 0 and 1.

The formulas 2.1 and 2.2 cover a forward pass of a single data point. The next step is to adjust the parameters w_i using a backward pass. The parameters w are learned by optimizing a loss function that measures the difference between the predicted probabilities and the actual binary labels in the training data. The most commonly used loss function for logistic regression is the cross-entropy loss, described in more detail in 2.1.9. The parameters w_i can be learned using optimization algorithms such as stochastic gradient descent, which iteratively updates the parameters in the direction of the negative gradient of the loss function with respect to w_i. Section 2.3 describes how these parameters are learned in more detail.

2.1.8 Performance Measures

As mentioned in section 2.1.7 on logistic regression, the model makes a prediction $\hat{y}_i$ where we can derive if the model predicts 1 or 0. This prediction happens both during training and evaluation. And since we have a labeled dataset containing the actual binary values, we can calculate different performance measures to determine the quality of the model. Many such performance measures are related to a confusion matrix. A confusion matrix is a table that summarizes the predicted and actual values for a binary classification task. The matrix has four cells, which represent the four possible combinations of predicted and actual values:

	Predicted Negative	**Predicted Positive**
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Table 2.1: A confusion matrix is a table that summarizes the performance of a classification model by showing the counts of true positive, true negative, false positive, and false negative predictions.

- True Positive (TP): The predicted class is positive and the actual class is also positive.
- False Positive (FP): The predicted class is positive but the actual class is negative.
- False Negative (FN): The predicted class is negative but the actual class is positive.
- True Negative (TN): The predicted class is negative and the actual class is also negative.

Using the confusion matrix, we can calculate various performance measures, such as accuracy, precision, recall, and F1-score [4]. The following formulas give them:

$$\begin{aligned} Accuracy &= \frac{TN + TP}{TN + FP + TP + FN}\,, \\ Precision &= \frac{TP}{TP + FP}\,, \\ Recall &= \frac{TP}{TP + FN}\,, \\ F1Score &= 2 * \frac{Precision * Recall}{Precision + Recall}\,. \end{aligned} \tag{2.3}$$

But what exactly do these performance measures mean? Consider the same example as previously, where the task is to train a model to classify movie reviews as positive or negative. Let's prefill the table with some numbers. The values in the confusion matrix are thoughtfully selected to demonstrate a specific scenario and emphasize a particular aspect.

	Predicted Negative	**Predicted Positive**
Actual Negative	20	10
Actual Positive	7	5

Table 2.2: Confusion matrix filled with some numbers. There are, in total, 42 predictions, and each prediction belongs to a particular cell.

Using the numbers from the confusion matrix, we can calculate the performance measures just described.

- Accuracy: This is the proportion of correctly classified instances out of the total number of instances. In this case, the accuracy is $\frac{20+5}{20+10+5+7} = 0.6$.
- Precision: This is the proportion of true positive predictions out of all positive predictions. In this case, the precision is $\frac{5}{5+10} = 0.33$.
- Recall: This is the proportion of true positive predictions out of all positive instances. In this case, the recall is $\frac{5}{5+7} = 0.42$.
- F1-score: This is the harmonic mean of precision and recall and is often used to summarize the classifier's performance. In this case, the F1-score will be $2 * \frac{0.33*0.42}{0.33+0.42} = 0.37$

This example highlights the importance of not always using (and trusting) the accuracy measure since it can be misleading. In the example, we have an accuracy of 0.6, whereas recall, precision, and F1-score are drastically lower. In scenarios where the dataset has an imbalanced portion of false positives and false negatives are not equal, precision and recall become essential measures to assess the model's performance.

Consider using these performance measures for a document ranking task. None of them consider the ordering of documents; therefore, they aren't helpful for document ranking tasks. In document ranking, the goal is to retrieve the most relevant documents for a given query and present them in a ranked order.

While traditional performance measures focus on the correctness of individual predictions, document ranking requires a more nuanced approach that considers the position of relevant documents in the ranking. For example, a ranking that places relevant documents at the top of the list is more valuable than one that places them at the bottom, even if both rankings have the same precision and recall.

Therefore, performance measures such as Mean Average Precision (MAP) are used in document ranking. MAP considers each document's relevance in ranking and weighs the importance of documents according to their position. MAP also accounts for users who often only examine the top few documents in the ranking. It is beneficial when the goal is to return a ranked list of relevant documents in response to a user's query.

The subsequent equations establish the definition of MAP:

$$AP(q) = \frac{1}{GTP} \sum_{k=1}^{D} P(k) * rel(k) \tag{2.4}$$

$$MAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(q) \tag{2.5}$$

In Equation 2.5, Q indicates the total number of queries. In equation 2.4, GTP refers to ground truth positives, D indicates total number of documents for the particular query, $P(K)$ refers to the precision at cut-off k in the document list and $rel(k)$ refers to the relevance at rank k. The $rel(k)$ function returns the value 1 if the document at rank k has an actual label = 1 and 0 otherwise. The $P(k)$ is the same precision formula explained in 2.3 with the difference of only looking at the top k results.

To understand MAP, we first need to understand Average Precision AP. Suppose a search engine retrieves documents for a particular query. The search engine retrieves ten documents, of which four are actually relevant to the query. The documents are ranked based on their relevance score, with the most relevant document at the top.

To calculate Average Precision (AP), we first calculate the $P(k)$ for each relevant document in the ranking given the document's ranking k. AP is the average of all documents' $P(k)$ values multiplied by the relevance function. Precision is the number of relevant documents retrieved at a given rank divided by the total number of documents retrieved at rank k. For example, the precision at rank 4 is 3/4 because we have three positive predictions at level $k = 4$, and the total number of documents is 4.

k	**Document**	**True Label**	**P(k)**	**P(k)*rel(k)**
1	Doc1	1	1/1	1
2	Doc2	1	2/2	1
3	Doc3	0	2/3	0
4	Doc4	1	3/4	3/4
5	Doc5	0	3/5	0
6	Doc6	0	3/6	0
7	Doc7	1	4/7	4/7
8	Doc8	0	4/8	0
9	Doc9	0	4/9	0
10	Doc10	0	4/10	0

Table 2.3: In this table, we elaborate the calculations needed to calculate the AP for a handful of ranked documents. Assume that an ML model has ranked the document in order of k and that each document has a true label. The right-most column highlights that the score is reduced if documents with true labels are placed at a lower stage than false label documents.

Now that we have calculated every $P(k) * rel(k)$ values, we can complete the formula, given in 2.4 to retrieve the AP for a particular query:

$$\begin{aligned} GTP &= 4\,, \\ AP = AP &= \frac{1 + 1 + \frac{3}{4} + \frac{4}{7}}{4} = 0.83\,. \end{aligned} \tag{2.6}$$

Then this calculation needs to be done for all queries along with its documents and the ranking models' prediction. Assume that we calculated AP for three queries and calculated the AP score with the results $AP_1 = 0.83$, $AP_2 = 0.79$, and $AP_3 = 0.90$. The MAP score is obtained by calculating the mean of all individual scores:

$$MAP = \frac{0.83 + 0.79 + 0.90}{3} = 0.84\,. \tag{2.7}$$

2.1.9 Binary Cross-Entropy Loss

Binary cross-entropy loss is a popular loss function used in binary classification tasks to measure the dissimilarity between predicted and actual probability distributions. It is also referred to as binary log loss.

The binary cross-entropy loss function measures the difference between the predicted probability distribution and the true probability distribution. In the case of binary classification, we want to predict the probability of the positive class (i.e., class 1) given an input instance. The output of a neural network for binary classification is often the probability of the positive class. This output value can be seen as a probability distribution over the two classes: the probability of the positive class and the probability of the negative class (i.e., class 0).

Suppose we have a binary classification problem where the labels are either 0 or 1, and we have a single output neuron that predicts the probability of the positive class. Let $\hat{y}$ be the predicted probability of the positive class for an input instance, and y be the true label (either 0 or 1) for the same input instance. The binary cross-entropy loss function is defined as follows:

$$BCE = -\frac{1}{N} \sum_{i}^{N} y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)$$

where N is equal to the total number of data points. The first term $(y * log(\hat{y}))$ measures the loss when the true label is positive (i.e., y=1). We want the predicted probability $\hat{y}$ to be as close to 1 because the true label is 1. Therefore, if the predicted probability is close to 1, the loss will be close to 0, and if the predicted probability is close to 0, the loss will be large.

The second term $((1-y)*log(1-\hat{y}))$ measures the loss when the true label is negative (i.e., y=0). We want the predicted probability $\hat{y}$ to be as close to 0 as possible because the true label is 0. Therefore, if the predicted probability is close to 0, the loss will be close to 0, and if the predicted probability is close to 1, the loss will be large.

The binary cross-entropy loss function is a smooth and continuous convex function in $\hat{y}$. Therefore, it is suitable for optimization using gradient-based methods such as SGD and other gradient descent-based optimization methods. During training, the model updates its parameters to minimize the binary cross-entropy loss, encouraging the model to make better predictions on unseen data.

2.1.10 Cosine Similarity

Cosine similarity is a mathematical measure that calculates the similarity between two vectors. It is a widely used technique used in various fields of ML. To compute cosine similarity, we first normalize the vectors to have unit lengths and then calculate the dot product of the two vectors, dividing it by the product of their Euclidean lengths. This results in a value between -1 and 1, where 1 indicates complete similarity, 0 indicates no similarity, and -1 indicates complete dissimilarity, allowing us to compare and measure the similarity between different data points.

More mathematically, cosine similarity between two numerical vectors is calculated using the following formula:

$$cosine\_similarity(A, B) = \frac{A \cdot B}{||A|| * ||B||} \tag{2.8}$$

where $\cdot$ represents the dot product operation, $||A||$ represents the Euclidean length or magnitude of vector A, and $||B||$ represents the Euclidean length or magnitude of vector B.

The formula for calculating the dot product operation between two vectors A and B is done by taking the sum of the element-wise products of the two vectors. The Euclidean length or magnitude of a vector A is computed as the square root of the sum of the squared elements of the vector.

$$A \cdot B = \sum_{i}^{d} (A_i * B_i) \tag{2.9}$$

$$||A|| = \sqrt{\sum_{i}^{d} A_i^2} \tag{2.10}$$

where A_i and B_i represent the i-th element of vectors A and B, respectively. d is the total number of elements in the vector.

For example, suppose we have two numerical vector representations, one for the word *cat* and another for the word *dog*. We can compute their cosine similarity using the formula described above. If the cosine similarity of the two vectors is close to 1, it suggests that *cat* and *dog* are similar in meaning, likely due to their shared attributes as household pets. If the cosine similarity is closer to 0, it suggests that *cat* and *dog* are less similar in meaning. Finally, suppose the cosine similarity is close to -1. In that case, it suggests that *cat* and *dog* are highly dissimilar in meaning.

2.1.11 Gestalt Pattern Matching

Gestalt pattern matching is a concept in psychology that describes how our brains naturally group visual elements into meaningful patterns or wholes. It's based on the idea that we perceive objects as a whole rather than a collection of individual parts.

When comparing two strings using Gestalt pattern matching, we can apply the same principles by looking at the overall shape and structure of the words rather than just the individual letters. In other words, we can try to match the "gestalt" of one string to the other.

To calculate the similarity between two strings, you count the number of matching characters, the longest common substring, plus any matching characters on either side

of that substring. Then multiply that number by two and divide it by the number of characters in both strings.

Let's say we have two strings, A and B. We can represent each string as a set of characters:

$$\begin{aligned} A &= \{a_1, a_2, ..., a_x\}, \\ B &= \{b_1, b_2, ..., b_y\}. \end{aligned} \tag{2.11}$$

The similarity of two strings is defined as:

$$D = \frac{2K_m}{x + y}. \tag{2.12}$$

where the similarity metric, D is given a value between zero and one. Closer to one indicates a higher string similarity. K_m is the number of matching characters.

For example, let's say we have the strings *farmvill* and *faremvie*. We can represent these strings as sets of characters:

$$A = \{"f", "a", "r", "m", "v", "i", "l", "l"\}$$
$$B = \{"f", "a", "r", "e", "m", "v", "i", "e"\}$$

We can then calculate the gestalt pattern similarity like this:

$$D = \frac{2 * (|\{"f", "a", "r"\}| + |\{"m", "v", "i"\}|)}{x + y} = \frac{2 * (3 + 3)}{8 + 8} = 0.75$$

2.2 Natural Language Processing

Natural Language Processing (NLP) is a field of study that deals with the interaction between computers and human language. It involves using computational methods and algorithms to process, analyze, and generate human language in written and spoken forms. NLP is highly interdisciplinary, drawing on computer science, linguistics, mathematics,

psychology, and other disciplines. We will look into NLP concepts related to the ML field and mainly focus on the language model BERT.

NLP and ML are often used in various applications, as ML provides a robust set of techniques for training models to recognize patterns in language data. Examples of such applications are Sentiment Analysis, Machine Translation, Text Classification, and Question Answering.

2.2.1 Data Cleaning

Data cleaning and pre-processing are crucial steps in NLP ML projects. It involves transforming raw text data into a clean and usable format that can be input into an ML algorithm. Reasons to do data cleaning and pre-processing could be improving performance by reducing noise and handling missing data. There are several aspects to consider when cleaning and pre-processing to transform the data to a more suitable format. You may encounter over-cleaning, where you remove important information or alter the meaning of the text, resulting in a lack of performance. Another aspect to consider is complexity and ambiguity. Natural language may use complex combinations of words to express its meaning; some words may express different meanings in different contexts. It can therefore be difficult to identify and standardize all the variations of the text combinations. Bias can also be introduced if, for example, a decision to remove or change certain words or phrases influences the outcome of the ML algorithm.

There are several standard methods for data cleaning in NLP. The first is stopword removal. Stopwords are common words in a language that is not informative for analysis, such as *the*, *and*, and *of*.

Stemming and lemmatization are other techniques to reduce inflected words to their base or root form. Stemming involves stripping the suffixes from words to reduce them to a standard stem. For example, the word *jumping*, after stemming, would result in the word *jump*. Lemmatization involves mapping words to their base form. For example, the word *were* would be turned into *be* after lemmatizing it.

Removing special characters and punctuation is beneficial and will sometimes increase performance. This can be done using regular expressions or string manipulation. Although, removing these may also result in a lack of performance if these tokens are informative for the particular task.

2.2.2 Tokenization

Word tokenization is breaking a piece of text into individual words or tokens and is a fundamental step in NLP. It is typically performed as a pre-processing step to prepare text data for further analysis or ML algorithms. A series of techniques have been proposed to achieve better performance on model when tokenizing raw text as input. Some techniques include whitespace tokenization and subword tokenization. Whitespace tokenization is the most basic one, where the algorithm only looks for whitespaces in the text, and all characters between the spaces will result in a token.

WordPiece tokenization is a subword tokenization technique used in modern language models. It creates a vocabulary of the most frequent subwords in the training corpus using an algorithm called Byte Pair Encoding (BPE). The subwords are obtained by iteratively merging pairs of adjacent characters or subwords based on their frequency in the corpus until a predefined vocabulary size is reached. Here is an example of how BPE works. Remember that BPE uses the whole training set and will typically produce a richer vocabulary than the one I create in this example. Consider that the following sentence is our entire corpus:

it is boring to see the previous model pretraining with a slow training speed

Then we find the most frequent pair of consecutive words. We can easily see that the adjacent character combination *ing* occurs three times. We can also notice that the character combination *pre* and *train* occurs two times. Our corpus will thus include the following tokens: *it, is, bor, ##ing, to, see, the, pre, ##vious, model, ##train, with, a, slow, train, speed*. Notice that the word *previous*, *pretraining*, and *training* is not in the corpus at all since we recreate them with the subwords in the corpus. The hashtag in front of some of the words indicates that these words are subwords and proceed by other words. We intentionally omit subsequent hashtags for the starting subword to treat it as a standalone word. For example, the subword *pre* will be treated the same as if *pre* occurred alone.

We can tokenize new texts now that we have used the BPE technique to create the vocabulary. The input text is first split into words using a whitespace tokenizer. Each word is then further split into subwords using the WordPiece vocabulary. If a word is absent in the vocabulary, it is split into individual characters. Tokenizing our sentence

will thus result in the following tokens: [it, is, bor, ##ing, to, see, the, prev, ##ious, model, pre, ##train, ##ing, with, a slow, train, ##ing, speed]

The WordPiece tokenization scheme has several advantages over basic tokenization techniques. It can handle out-of-vocabulary words by breaking them into subwords, improving the model's performance on rare or unseen words. It also allows the model to learn meaningful representations of morphologically rich languages, where words can have many inflected forms and word boundaries are not always clear. It can also handle misspellings since it can break down the words into subwords where the subwords can be regarded as the same.

2.2.3 Word Embeddings

Word embeddings are distributed representations for words that capture the semantic and syntactic similarity between words. They are dense vectors or matrices trained from large text corpora using neural network-based models.

Before explaining word embeddings, let's look at One-hot vectors. One-hot vectors represent words as sparse one-hot vectors. The vector size is typically set to the number of tokens in the vocabulary, sometimes up to 30 000 or even 50 000 tokens. A one-hot vector representation for a particular token will have a single 1 value at the word's vocabulary position and all other values 0. This vector can then be used as input to a model. Word embeddings are another way of representing words by representing them as continuous vectors in a low-dimensional space to capture the semantic meaning of words. The size of these vectors is drastically smaller than the size of the vocabulary and is much richer in density. Word embeddings are usually trained in a fashion resulting in similar words being located close to each other in the embedding space. Their relations and similarities can be computed using mathematical operations, such as cosine similarity, explained in 2.1.10.

One of the advantages of word embeddings is that they can be pre-trained on large text corpora and then fine-tuned on smaller task-specific datasets, saving computational time and resources. However, word embeddings come with some limitations. One limitation is that they are context-independent and may not capture the nuances and variations of word meanings across different contexts and domains. This can lead to biases and errors in downstream tasks. Various methods have been proposed to address this issue by incorporating contextual information, such as contextualized word embeddings, to

capture words' rich and diverse meanings in different contexts and domains. In section 2.4.6, we will describe how BERT handles this problem by creating contextualized word embeddings. Also, Word embeddings cannot represent phrases, sentences, or documents as embeddings and are thus limited to only representing single tokens or words.

2.2.4 Term Frequency–Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical method that reflects a term's importance to a document within a collection of documents. It is commonly used in NLP for text classification, information retrieval, and other tasks that involve processing and analyzing text. In this research, I will use TF-IDF to train a logistic regression model for binary classification.

TF-IDF is calculated by multiplying Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures the frequency of a term in a document. It is calculated by dividing the number of times a term appears in a document by the total number of terms in the document. The resulting value is often normalized to prevent bias towards longer documents.

$$TF(t,d) = \frac{\text{Number of occurrences of term t in document d}}{|d|} \tag{2.13}$$

where $|d|$ is the total number of terms in document d. We can consider TF as a percentage of a term's frequency in a given document. Now why can't we use TF alone and skip the IDF? Usually, a particular document will contain many connection words like *the*, *a*, and *of*, which are included for connecting more meaningful words. So these words will get a higher TF score than the special words we are more interested in since the connection words occur more often.

IDF measures a term's significance in the entire collection of documents (training set). It is calculated by taking the logarithmic function of the total number of documents in the corpus divided by the number of documents that contain the term.

$$IDF(t) = \log\left(\frac{|D|}{\text{Number of documents with term t}}\right) \tag{2.14}$$

where $|D|$ is the total number of documents. The TF-IDF score for a term t in a document d is calculated by multiplying the TF and IDF values.

$$TF\_IDF(t, d) = \text{TF}(t, d) * \text{IDF}(t) \tag{2.15}$$

The score will be higher when a term frequently appears in a document (high TF) and is rare in the collection of documents (high IDF), indicating that the term is more significant and relevant to the document.

Let's try to understand TF-IDF with an example. Consider that our dataset contains the following documents:

- Document 1: *A randomized, double-blind placebo-controlled clinical trial with penicillin V in general practice*
- Document 2: *Prescription strategies in acute uncomplicated respiratory infections. A randomized clinical trial*
- Document 3: *A randomized controlled trial of antibiotics on symptom resolution in patients with a sore throat*

For this example, we split the document by spaces and consider all characters between the splits as terms. The following calculates the TF-IDF score for the term *randomized* in Document 2:

$$TF(\text{randomized, Document 2}) = \frac{1}{11}$$
$$IDF(\text{randomized}) = log(3/3) = 0$$
$$TF\_IDF(\text{randomized, Document 2}) = \frac{1}{11} * 0 = 0$$

Looking at the documents at hand, we can see that we will get a TF-IDF score equal to 0 for the term *randomized* in all documents because the IDF score is 0. But what would the TF-IDF be for the word *respiratory* in document 2?

$$TF(\text{respiratory, Document 2}) = \frac{1}{11}$$
$$IDF(\text{respiratory}) = log(3/1) = 0.477$$

$$TF\_IDF(\text{respiratory, Document 2}) = \frac{1}{11} * 0.477 = 0.043$$

Considering the whole document collection, we get a higher score for the term *respiratory* than the word *randomized* in document 2. This is because *respiratory* only appears in this document and not in any other, indicating a rare word. TF-IDF scores can be further used as features to train ML models.

2.2.5 Document Ranking

Document ranking is a task in information retrieval that involves ordering a set of documents based on their relevance to a given query.

Given a query and a collection of documents, the goal is to rank the documents according to their relevance to the query, with the most relevant document at the top of the list. This task is typically performed using a ranking model that assigns a score to each document based on its relevancy to the query.

In the early days of information retrieval, document ranking relied on simple keyword matching and frequency of query terms. Term weighting and TF-IDF were introduced to improve ranking. ML methods expanded the range of features considered. Deep learning, including transformer models, recently revolutionized document ranking, achieving state-of-the-art results by learning complex representations and handling large datasets.

The task of using ML to train document ranking algorithms is called the Learning To Rank (LTR) framework. The following explanation of using the LTR framework to train document ranking algorithms is based on the article *Neural ranking models for document retrieval*[20]. LTR is based on the observation that relevance is a subjective and complex concept that depends on various factors, such as the user's context, preferences, and information needs. Therefore, LTR aims to learn a ranking model to capture these factors and provide personalized and context-sensitive document ranking.

We divide the different LTR methods into pointwise, pairwise, and listwise. In pointwise methods, each document is treated as a single data point, and the model is learned to predict the relevance score of each document independently. This means the model takes a document and a query as input and predicts a relevance score as output. The most commonly used pointwise method is the logistic regression-based method, which involves learning a regression model that maps the features of a document and query to a relevance

score. In pairwise methods, each pair of documents is treated as a single data point, and the model is learned to predict the relative order of two documents. This means that the model takes two documents and a query as input and predicts which of the two documents is more relevant to the query. Pairwise methods require labeled data containing pairs of documents and corresponding relevance judgments. In listwise methods, the model is learned to optimize the ranking of an entire list of documents. Thus this method requires labeled data containing entire lists of documents and relevance judgments.

Things to be aware of using the pointwise training approach is that it does not consider the relative order of the documents. It may not capture the nuanced differences in relevance among documents or ordinal or continuous relevance labels. It can also be sensitive to class imbalance, where the number of relevant and non-relevant documents vastly differs.

2.3 Neural Network

In this section, we delve into Neural Networks, a powerful class of ML models inspired by the human brain. We explore their architecture, functioning, and how they can effectively learn from data to solve complex tasks.

2.3.1 Basics of Neural Networks

Neural networks are a type of ML model that consists of layers of interconnected nodes, or neurons, that process input data and generate output predictions.

The basic building block of a neural network is a neuron, which takes one or more input values, multiplies them with a numerical number (further referred to as a weight), summarizes the individual results, and applies an activation function to the result. The activation function is a non-linear function that introduces non-linearity into the network and allows it to model complex relationships between the input and output variables. Section 2.3.5 explains activation functions in more detail.

The goal of training the model is to minimize the difference between the predicted and actual values. And how can we achieve this? Neural networks are typically trained using a gradient descent-based optimizer, explained in more detail in section 2.3.2 and

2.3.3, which updates the parameters of the network to minimize a loss function. By minimizing the loss function, we aim to find the optimal set of parameters to predict the output data with the smallest possible error. Minimizing the loss function is correlated with minimizing the difference between the predicted and true values because the loss function measures this difference.

The architecture of a neural network can vary widely depending on the task it is designed to solve. Some common types of neural networks include Feedforward neural networks (FFNN), Convolutional neural networks, and Recurrent neural networks. We will mainly focus on FFNN in this research. FFNN is the simplest type of neural network, where the information flows in one direction, from the input layer through one or more hidden layers to the output layer.

FFNNs are widely used for supervised learning tasks such as classification and regression. They consist of one or more layers of neurons that process the input data and generate output predictions. Every FFNN has an input layer, a series of hidden layers, and an output layer.

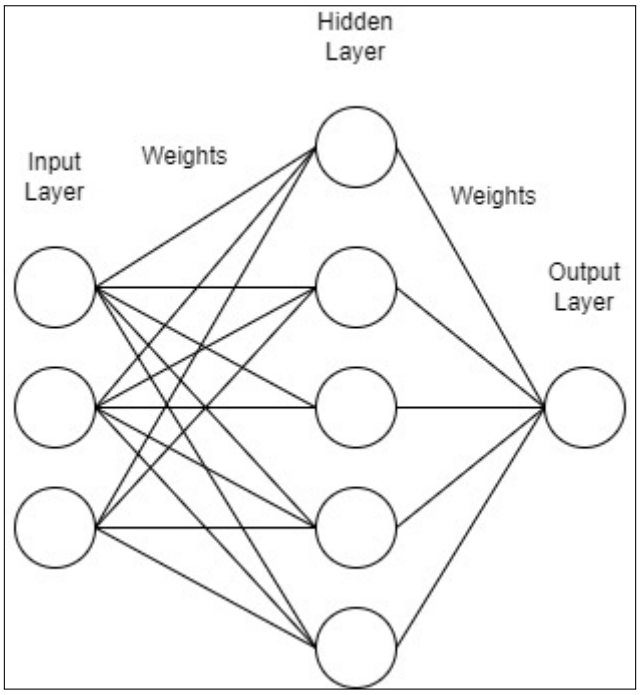

Figure 2.3: Illustration of a feed-forward network. The computational flow goes from the input layer to the hidden layer and finally to the output layer. The lines between two neurons are a single weight. The output value from one neuron is multiplied by this weight and then passed on. If a neuron has more than one line as input, the values from each line are summed together before an activation function is applied.

Figure 2.3 shows the basic building block of a feedforward neural network. The circles represent neurons that take one or more input values, multiply them by weight (the black lines), sum all values, and then apply an activation function to the result. The neuron's output is then passed as input to the next layer of neurons. FFNNs typically consist of an input layer, one or more hidden layers, and an output layer. The input layer takes the raw input data, such as a sequence of text, and passes it to the first hidden layer. Each subsequent hidden layer processes the previous layer's output using a set of learned weights and biases and applies an activation function to the result. The output layer generates the final prediction based on the output of the last hidden layer.

In a neural network, the weights and biases determine the strength of the connections between neurons. The network weights are learned during the training process and are used to adjust the behavior of the network to achieve better performance on a given task. The weights in a neural network are typically represented as matrices or vectors, depending on the architecture of the network. For example, in a fully connected feed-forward neural network, the weights between the input layer and the first hidden layer are represented as a matrix. Each matrix element corresponds to the weight between a specific input neuron and a specific hidden or output neuron.

2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization technique that optimizes an objective function. It is a stochastic optimization algorithm that estimates the gradient of the loss function based on a random subset of the training data. SGD iteratively updates the model parameters by taking small steps in the direction of the negative gradient of the loss function.

Let's consider a simple example of using SGD to minimize the loss function of a logistic regression model. Suppose we have a dataset of n training examples $(x_1, y_1), ..., (x_n, y_n)$, where x_i is the input feature vector and y_i is the corresponding binary class label (either 0 or 1). We want to train a binary classification model using logistic regression to predict the probability of an input belonging to class 1. Keep in mind how we calculated z in logistic regression, ref 2.1.7 where z was equal to the sum of element-wise multiplication of weights and feature values. $\hat{y} = p(y = 1|x)$ is $sigmoid(z)$, meaning that we map the output z to a value between 0 and 1, now representing a probability. $J(w)$ is the training data's negative log-likelihood and logistic regression loss function for a binary classification task. Stochastic gradient descent aims to find the optimal values of the

weights, w, that minimizes the loss function $J(w)$. Let's define how the algorithm work when considering just a single data point. Usually, there is always a bias term, b, which is an additional neuron and is usually represented as a constant value instead of a learnable weight. We will only focus on the updating the weights, w, in this example

1. Initialize the model parameters w to random values. For t from 1 until converge.
2. Choose (x_i, g_i) at random and compute the gradient of the loss function concerning the model parameters w, which is given by:

$$g_t = (p(y = 1|x_i) - y_i) * x_i$$

3. Update the model parameters using the gradients and a learning rate α, which controls the size of the steps taken in the direction of the negative gradient:

$$w_t = w_{t-1} - \alpha * g_t$$

4. Repeat steps 2 and 3 until the loss function converges or a maximum number of iterations is reached. The t parameter increases iteratively throughout the training sequence.

Steps 2 and 3 are referred to as backpropagation. Step 2 is computing the gradients of the loss function for a single data point. Step 3 is optimizing the weights by subtracting the gradients multiplied with a learning rate α from the original weights. We are thus updating the weights to provide a function with minimum loss.

In each iteration of SGD, the gradients are estimated based on a random subset of the training data, which introduces stochasticity into the optimization process. This randomness helps to prevent the algorithm from getting stuck in local minima and can improve the model's generalization performance.

2.3.3 ADAM Optimizer

The Adam optimizer is a popular SGD optimization algorithm. The name *Adam* stands for *adaptive moment estimation*, which refers to the optimizer adapting its learning rate based on the parameters' past gradients and second moments. To fully understand

ADAM, let's first look at other optimization techniques based on SGD, namely SGD with momentum and RMS.

SGD with momentum is a technique used to accelerate the convergence of the optimization process. Momentum is a moving average of the gradients that accumulates the past gradients in the current gradient's direction, smoothing out the variations in the gradient descent process and helping to overcome local minima. Keeping in mind that the weight updating phase was given by $w_t = w_{t-1} - \alpha * g_t$ in 2.3.2, where α is the learning rate, g_t is the gradient of the loss function and t is the current iteration step. SGD with momentum is different from SGD in the way it uses the moving average of the gradients instead of only the gradient of the current step. The weight update will therefore look like this:

$$\begin{aligned} m_t &= \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \,, \\ w_t &= w_{t-1} - \alpha * m_t \,. \end{aligned} \tag{2.16}$$

where β_1 is the momentum coefficient, typically set to a value between 0.8 and 0.99. The momentum term $\beta_1 * m_{t-1}$ acts as a kind of inertia, which helps the optimizer to smooth out the variations in the gradient descent process and overcome sticking in local minima.

Root Mean Square (RMS) is another gradient descent optimization. It differs from SGD in that it adjusts the learning rate of each parameter based on the magnitude of its gradients. RMS also considers the recent history of the gradients to improve the optimization process further. This is achieved with the following formulas:

$$\begin{aligned} v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \,, \\ w_t &= w_{t-1} - \frac{\alpha}{\sqrt{v_t + \epsilon}} * g_t \,, \end{aligned} \tag{2.17}$$

where β_2 is the decay rate, which is typically set to a value between 0.9 and 0.999, g_t^2 denotes element-wise squaring, and ϵ is a small constant added for numerical stability. The denominator $\sqrt{v_t + \epsilon}$ scales the step size according to the magnitude of the gradients so that large gradients result in smaller steps and small gradients result in more giant steps.

Now that we know how SGD with momentum and RMS works, we can move on to the ADAM optimizer, a combination of these two. In each iteration step t, the following variables are calculated:

$$\begin{aligned} m_t &= \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \,, \\ v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \,, \end{aligned} \tag{2.18}$$

where β_1 and β_2 are the exponential decay rates for the first and second moments, respectively, and g_t^2 denotes element-wise squaring. We then calculate the bias-corrected estimates of the first and second moments:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \,, \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \,. \end{aligned} \tag{2.19}$$

Finally, we update the parameters w_t using the bias-corrected estimates of the moments:

$$w_t = w_{t-1} - \alpha * \frac{\hat{m}}{\sqrt{\hat{v}_t + \epsilon}} \tag{2.20}$$

where ϵ is a small constant added for numerical stability, $\sqrt{\hat{v}_t + \epsilon}$ denotes element-wise square root. The Adam algorithm combines the ideas of momentum and RMS by calculating the first and second moments of the gradients and then updating the parameters based on the bias-corrected estimates of these moments.

2.3.4 Linear Schedule Warmup

A linear schedule warmup is used in training deep learning models that adjust the learning rate over time. In particular, it is used to gradually increase the learning rate at the beginning of training before allowing it to decrease later.

The basic idea behind a linear schedule warmup is that at the start of training, the model's parameters are likely to be far from optimal, and the learning rate needs to be high enough to ensure that the model can rapidly converge toward a better solution.

However, if the learning rate is set too high initially, the model may diverge or oscillate around the solution, slowing the learning process.

To balance these concerns, a linear schedule warmup gradually increases the learning rate from a small initial value to a higher target value over a fixed number of training steps. This period of gradual increase allows the model to explore a larger space of solutions and helps it find a good starting point for optimization. Once the warmup period is complete, the learning rate is allowed to decrease gradually, using a standard schedule or decay function.

The length of the warmup period and the target learning rate can be adjusted depending on the specific problem and architecture being used.

2.3.5 Activation Functions

An activation function is a non-linear function that is applied to the output of a neuron in a neural network. The activation function introduces non-linearity into the network and allows it to model complex relationships between the input and output variables. Without an activation function, a neural network would be a linear function of its inputs, limiting its ability to model non-linear relationships.

Different activation functions are used in neural networks, each with advantages and disadvantages. We have already described Sigmoid, but others exist, such as Rectified Linear Unit (ReLU) and Gaussian Error Linear Unit (GELU). ReLU is one of the most popular activation functions used in neural networks and is defined as $f(x) = max(0, x)$ and looks as follows:

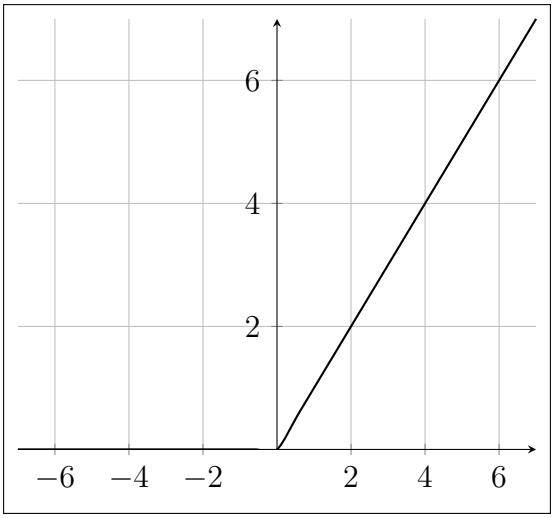

Figure 2.4: ReLU function plot illustrating the commonly used activation function in neural networks, where input values below zero are mapped to zero, while positive values remain unchanged, facilitating non-linearity and feature extraction in deep learning models.

ReLU has the advantage of being simple and computationally efficient, and it has been shown to work well in practice. It also has the desirable sparsity property, meaning that only a subset of the neurons will be active for any given input, which can help reduce overfitting.

The Gaussian Error Linear Unit (GELU) is an activation function that has been demonstrated to be effective in deep neural networks. It computes the product of its input and the cumulative density function of the normal distribution at that input. The error function is frequently employed to calculate the cumulative distribution function of a Gaussian. Therefore, GELU is defined as the Gaussian Error Linear Unit using the following approximation [32]:

$$GELU(x) = 0.5 * x * \left(1 + tanh\left(\frac{x}{\sqrt{2}}\right)\right)$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

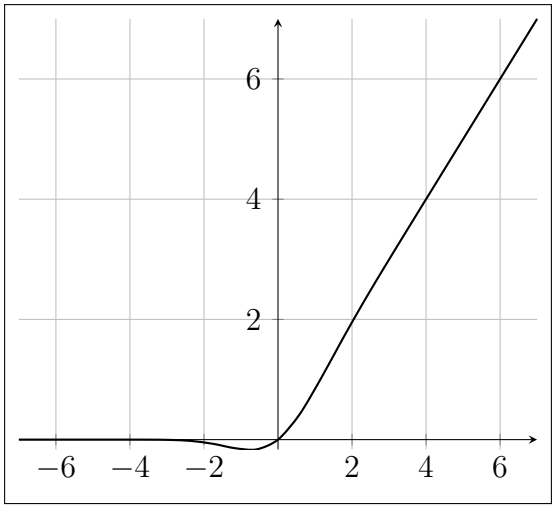

Figure 2.5: GELU function plot illustrating the Gaussian Error Linear Unit activation function, which smoothly approximates the identity function for small input values and introduces non-linearity for larger input values, enhancing the expressive power of neural networks in various machine learning tasks.

GELU has the advantage of being smooth and continuous, which can help improve the gradient flow during training. It also has a similar sparsity property as ReLU but with a smoother transition between active and inactive neurons.

2.3.6 Backpropagation

Section 2.3.2 explains how weights are updated for a single datapoint for a single linear layer. When considering backpropagation in neural networks, the concept of chain rule refers to how the gradients of the loss function propagate backward through the network. When performing backpropagation, the gradients are calculated with respect to the parameters (weights) of the network. These gradients indicate the direction and magnitude of the adjustments required to minimize the loss function. The gradients are computed using the chain rule, which allows the error signal to flow backward from the output layer to the input layer.

Let's consider a simplified feedforward network with multiple layers to understand how the chain rule is propagated in backpropagation. Each layer consists of weighted connections, followed by an activation function. The network takes an input, propagates it through the layers, and produces an output. During the forward pass, the input signal is successively transformed by each layer's weights and activation functions until the final

output is generated. The intermediate outputs of each layer are stored for later use in the backward pass. In the backward pass, the chain rule comes into play. Starting from the final output layer, the derivative of the loss function with respect to the output is computed. This derivative represents the sensitivity of the loss function to changes in the output.

Loss scaling is necessary for deep neural networks because gradients decrease as they pass through multiple layers. This occurs because each layer multiplies the gradients, and this multiplication can cause the gradients to become very small. When small gradients propagate through the network, it can result in slower convergence or even vanishing gradients, which means they become too tiny to update the network's parameters effectively. One way to address the loss scaling issue is to use the specialized optimization algorithms, SGD with momentum, RMS, and ADAM to dynamically adjust each parameter's learning rate based on its historical gradients. These methods can effectively scale the gradients to improve convergence and training efficiency.

2.3.7 Dropout

Dropout is a generalization technique used in neural networks where we avoid using the output from all neurons at every forward pass. By not allowing all neurons to contribute in a forward pass, we avoid making the model learn the training data too well and force it to learn general patterns. Dropout is used by setting a parameter as a percentage of how many neurons to exclude at random. And which particular neurons to exclude varies from each forward pass. For example, we have a linear layer of 10 neurons where a dropout layer with a percentage parameter of 0.1 follows. This implies that 10% of the neurons, which is 1, will be excluded in a particular forward pass. Regarding which dropout parameter to choose, this usually has to be empirically tested and chosen using the models' performance on validation set [7][p. 112].

2.4 Transformers

The following section is described based on how Sudharsan Ravichandiran explains in the book *Getting Started with Google BERT* [28]. Also, the figures used throughout this section are from the same sources with the publisher's permission.

Transformers, introduced in the paper *Attention Is All You Need* [3], is one of the most prominent techniques used in NLP today. By showing more significant results, it has replaced Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) in many downstream tasks, such as machine translation and text generation. BERT is among the algorithms used for these tasks and utilizes the transformers technique. One of the problems with RNN and LSTM that transformers solve is catching long-term dependencies in sentences. The transformers are based on the attention mechanism, which is self-attention and multi-head attention. Let us delve further into the transformer model, exploring its underlying mechanisms and components in greater detail.

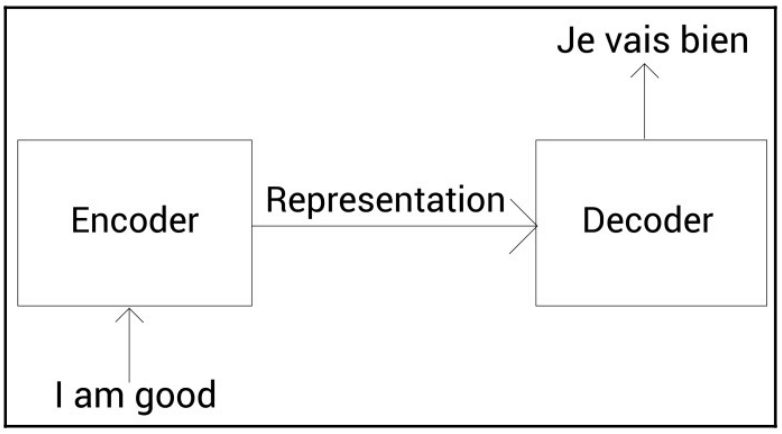

Figure 2.6: Illustration of Transformer Architecture. Transformers are built up of encoders and decoders. This figure shows the input and output interaction between them.

The transformer consists of encoders and decoders. As seen in Figure 2.6. The figure shows how the encoder takes an input sentence and outputs a sentence as a fixed representation. Then the decoder takes this representation generated by the encoder and generates a target sentence (a prediction). So if the task is to use the model for machine translation, you can input a raw sentence, and the transformer will output a predicted sentence on the language it is trained for.

2.4.1 Encoder

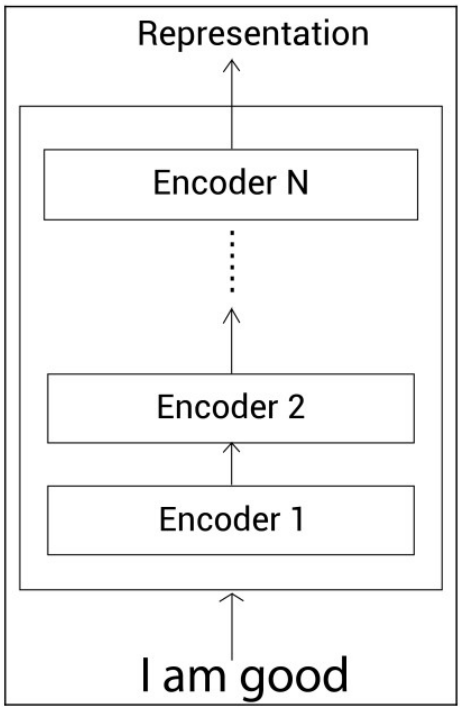

Figure 2.7: Encoders stacked on top of each other. The text is passed as input to the first encoder and its output is passed as input to the next encoder. The final output is a representation of the input sentence.

Figure 2.7 shows how encoders are stacked on top of each other in the transformer, where the output of one encoder is fed as input to the other. The same applies to the decoders. The encoder and decoder are essential components in a transformer and perform multiple mathematical operations. Let's first look at the encoder and its ability to transform the raw text into a representation of numbers. To understand this, we need to dive into the different components of the encoder, namely the multi-head attention and feedforward network.

2.4.2 Attention Mechanism

The attention mechanism is used to understand how much each word in a sentence relates to all the other words. The encoders use multi-head attention, which is a series of self-attentions. Let's first look at how a single self-attention works with an example. Considering the following sentence:

The man couldn't lift his son because he was so heavy

For a native-speaking English person, the preceding sentence is easy to understand. But it might not be easy to interpret for someone just learning English. In our case, it is an ML model we want to learn English from scratch. In the given sentence, multiple words can be interpreted in the wrong ways. Look at the pronoun *he*. It is unclear that it relates to the word *son*. This relation is due to the following word *heavy* indicating that the *son* is too *heavy* to be lifted. If we substitute the word *heavy* with *weak*, the word *he* suddenly changes relation and now relates to the word *man*. The grammatical structure of the sentence is still the same. For such a case, self-attention will help us out.

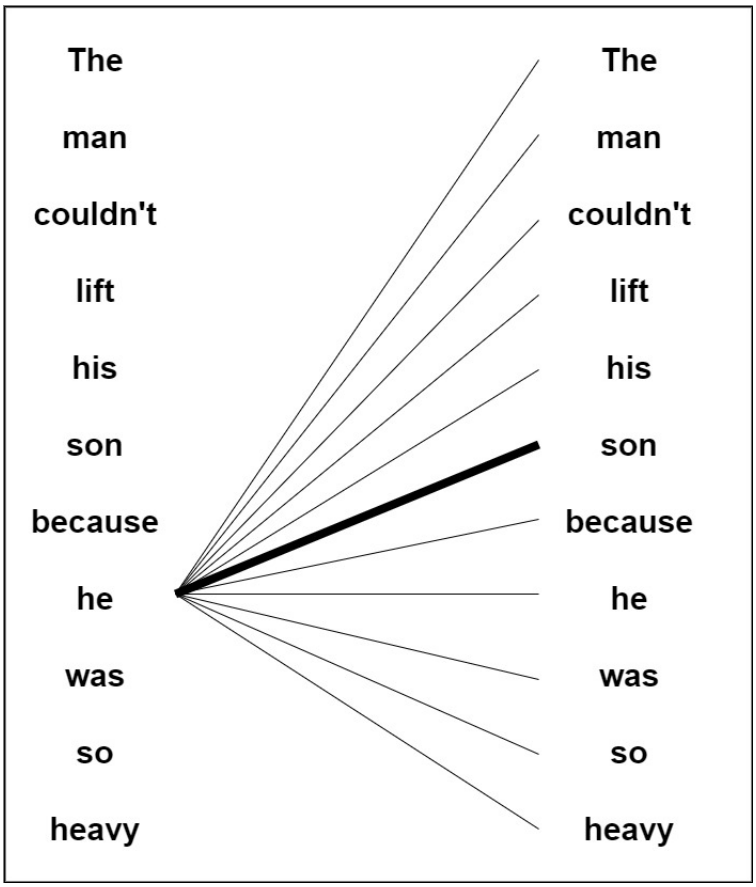

Figure 2.8: Example of the Self-attention mechanism for a particular sentence. The lines represent relational strength between the word *he* and all the other words in the sentence. You can see that the line between *he* and *son* is thicker than the others, indicating a stronger relationship between these two words in the sentence.

For a given sentence, we compute a numerical representation for each word. While computing this representation, the model computes a relation between each word and

all the other words in the sentence. So for our example-sentence, when computing the representation for the word *he*, the model also computes a relation between the word *he* and all the other words in the sentence, and thus also the word *son*. Hopefully, throughout the learning phase, the model will calculate a higher relation for this word pair than others in the sentence. Figure 2.8 highlights this more substantial relation between *he* and *son* with a bolder line than the others.

Now let's look at how this work in more detail. Given the input sentence *I am good*, first, we retrieve the word embeddings for each word. This is done using a lookup table where each word is mapped to a feature vector representation in the embedding matrix. Figure 2.9 shows an example word embeddings for the sentence *I am good*, where the first row in the matrix is the word embedding (vector) for the word *I* and so on. We can thus represent a sentence numerically by stacking up a series of word embeddings, resulting in a matrix.

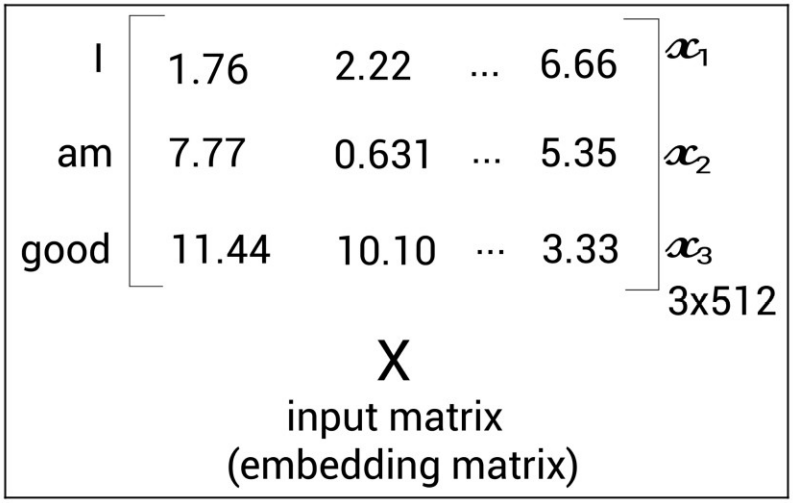

Figure 2.9: Sentence represented using word embeddings. For this example, the word embedding for each word is 512 numerical values, and since the input sentence contains three words, the input matrix has a size of 3x512. This matrix can be referred to as input matrix, embedding matrix, or just X.

Next, we are creating three new matrices, query, Q, key, K, and value, V. To create these three matrices, we need three additional weight matrices which contain trainable parameters. Each matrix is computed by multiplying the input matrix with the corresponding weight matrix.

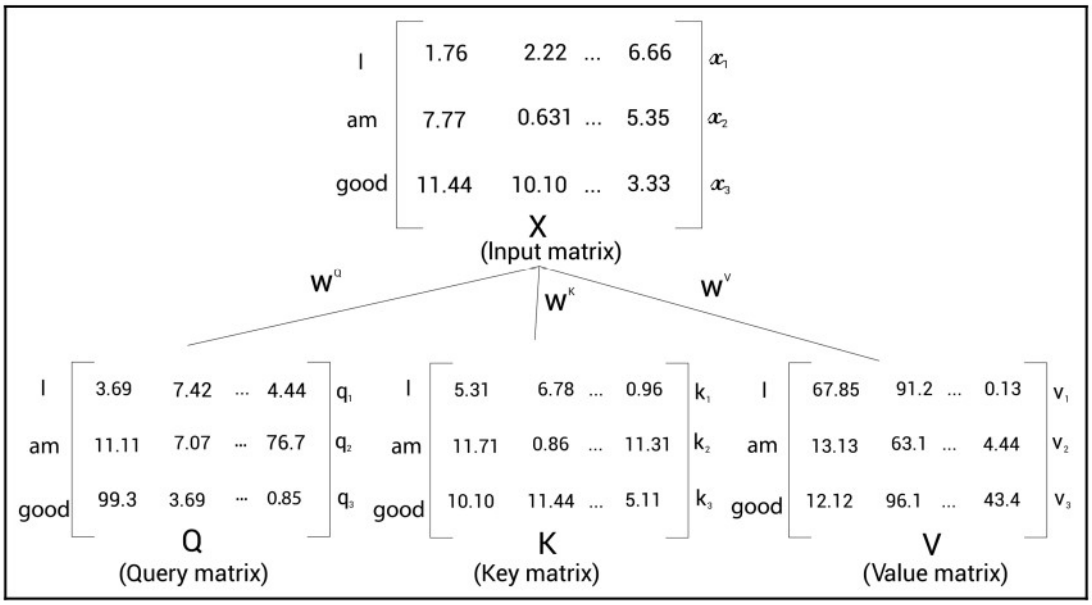

Figure 2.10: Creation of the Q, K and V matrices. Figure shows how the Q, K, and V matrices are created by multiplying the input matrix with the associated weight matrices W^Q, W^K and W^V

Figure 2.10 shows how this matrix multiplication is done. But why exactly are we doing this? How are these matrices used to calculate the self-attention of the input sentence? First, we need to calculate the words' similarity in the sentence. This is done by the formula QK^T, where we calculate the dot product between the Q matrix and transposed K matrix. In this operation, we calculate for each embedded representation of input words the dot product between the embedded representation of all other words. A dot product is used because it represents how similar two vectors are. Next, we need some operations to avoid exploding gradients. Therefore QK^T is divided by the square root of the dimension of the K matrix, $\sqrt{d}_k$. Thus, we have now calculated $\frac{QK^T}{\sqrt{d}_k}$.

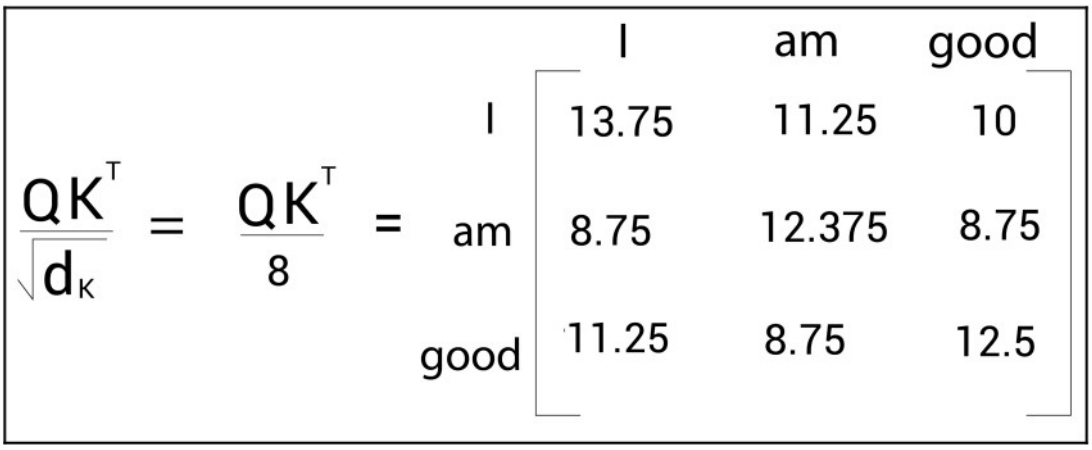

Figure 2.11: Illustration of word similarity matrix within a sentence, with scaling. We are calculating $\frac{QK^T}{\sqrt{d_k}}$ that results in the matrix on the figure.

Figure 2.11 shows the calculation of how similar each word in the sentence is to each other with a scaling factor. And that's great, but we don't know which values are referred to as high similarity or low similarity. There is thus a need to perform normalization. The next step is thus to apply a softmax operation on the entire matrix.

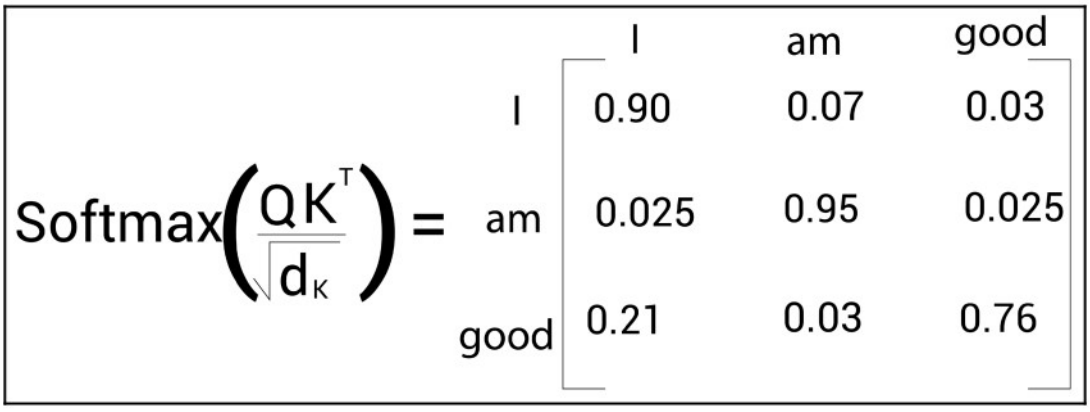

Figure 2.12: Illustration of word similarity matrix within a sentence, normalized. In more detail, we are calculating $softmax(\frac{QK^T}{\sqrt{d_k}})$, which is applying softmax on the matrix shown in Figure 2.11

Figure 2.12 shows how the softmax function has changed the values from 2.11 to range between 0 and 1 on each row. After softmax, the sum of each row can be summed to 1, meaning that each row contains similarity values for all words in the sentence. The highest value index represents the most similar word for the word associated with the given row.

To finish up the self-attention calculation, we multiply the newly calculated $softmax(\frac{QK^T}{\sqrt{d_k}})$ with the value matrix V. By doing this, we will get the sum of the value vectors in V, weighted by the scores calculated in $softmax(\frac{QK^T}{\sqrt{d_k}})$. For example, for the first word *I*, we will obtain the values from V weighted by how important the words are to the word *I*. Thus the final equation for calculating the self-attention z is:

$$z = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V\,. \tag{2.21}$$

Now let's move on to multi-head attention. Multi-head attention is self-attention multiple times. If we only use single self-attention, the impact of some words will be dominated by others. Look at Figure 2.12. The numbers in the matrix show that they only contain 7% from the word *am* and only 3% from the word *good*. This scenario is excellent for ambiguous words, meaning they can express multiple meanings (or refer to multiple words in the sentence). We will then have a clear idea of which other words it is related to. But for words that (almost) always express the same meaning, we want an accurate result and thus calculate multiple self-attentions and concatenate them.

$$MultiHeadAttention(Q, K, V) = Concatenate(Z_1, Z_2, ..., Z_i)W_0 \tag{2.22}$$

The concatenated value containing multiple self-attention matrices is multiplied with a new weight matrix, W^0, from Equation 2.22, which will be learned during training. The output from the multi-head attention, z, are then passed on to two densely connected feedforward layer. By densely connected, we mean that every neuron from the previous layer is connected to every neuron in the current layer. The multi-head attention and feedforward component uses the ReLU activation function on their outputs. Now that we know how the encoder learns how each word in a sentence relates to the other words, we need to know how the encoder considers each word's position.

2.4.3 Positional Encoding

The positional encoding is part of the encoder that learns the position of each word's impact on the whole sentence. As mentioned earlier, words are passed in parallel to the model, both for increasing training time and learning long-term dependencies. But how exactly can the encoder understand the meaning of a sentence if it doesn't know

the position of each word? The positional encoding handles this by adding element-wise addition information to the word embeddings. So before feeding the word embeddings to the multi-head attention component, we add a P matrix. The P Matrix is calculated using the following equations:

$$P(pos, 2i) = sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{2.23}$$

$$P(pos, 2i+1) = cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{2.24}$$

Equation 2.23 and 2.24 are the equations used to calculate the positional encoding matrix. In these equations, *pos* is the position of the words in the sentence, whereas i implies the position of the embedding for the word. For the particular sentence *I am good*, where I is the 0^{th} position, and so on. We can substitute the variables and will get the following matrix. We will thus get a unique number for a word on a particular position.

P =				
I	sin(0)	cos(0)	sin(0/100)	cos(0/100)
am	sin(1)	cos(1)	sin(1/100)	cos(1/100)
good	sin(2)	cos(2)	sin(2/100)	cos(2/100)

Figure 2.13: Example of P matrix

2.4.4 Add-And-Norm

The add-and-norm component in the encoder refers to a step in which residual connections and layer normalization are applied. After the multi-head self-attention and feed-forward neural network layers within an encoder, the output is combined with the input using an element-wise addition. This allows the model to preserve important information from

the input throughout the encoding process. Following the addition, layer normalization is performed to normalize the output and improve the stability and performance of the model. The add-and-norm operation aids in information flow and gradient propagation, contributing to the effectiveness of the transformer encoder.

2.4.5 Encoder summed up

Figure 2.14 sum up the main building blocks in the encoder. The illustration includes two encoders, and the first one is expanded. The text is passed as input to generate an embedded representation of the text. Then the positional encoding is applied by element-wise adding additional information. Then the embedding is passed on to a series of encoders, each applying multi-head attention and Add norm techniques.

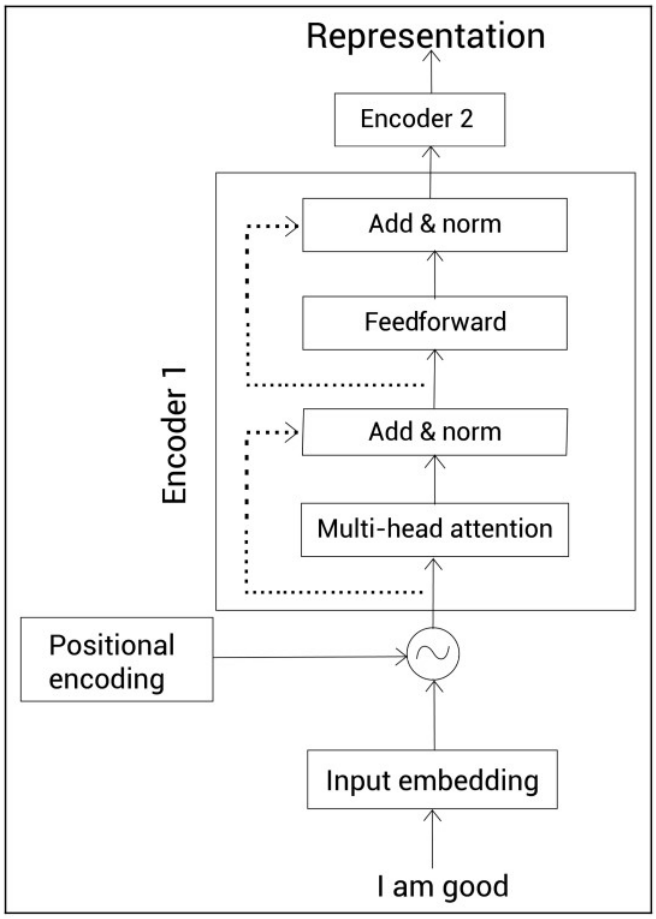

Figure 2.14: Illustration of the encoder architecture in a transformer model. The input sentence undergoes an initial embedding process, followed by the addition of positional encoding. This processed input is then fed into the first encoder, which includes a multi-head attention mechanism. Subsequently, the output passes through an add-and-norm component, followed by a feedforward neural network layer and another add-and-norm operation. The resulting output from encoder 1 serves as the input for encoder 2, enabling sequential encoding and enhancing the model's ability to capture complex relationships within the input sequence.

2.4.6 BERT

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model developed by Google. It is a deep learning model designed to generate high-quality representations of natural language text for a wide range of natural language processing (NLP) tasks. BERT is built on the Transformer architecture. See section 2.4 for more details.

Although BERT is built using the Transformer architecture, it only uses the encoder layers, not the decoder layers. In the Transformer architecture, the encoder layers take

as input a sequence of tokens and generate a sequence of hidden representations, which are then fed to the next layer. The final encoder layer produces the contextualized representations for each token in the input sequence.

In the case of BERT, the model is pre-trained on a large corpus of text using two unsupervised learning tasks known as masked language modeling (MLM) and next sentence prediction (NSP).

MLM aims to train the language model to predict missing words in a sentence by masking out a randomly selected portion of the input sequence and training the model to predict the missing words. During the MLM training task, a portion of the input tokens in a sentence is randomly selected and replaced with a particular [MASK] token. For example, in the sentence *I like to eat apples and oranges*, a portion of the tokens might be masked as *[MASK] like to eat [MASK] and oranges*. The masked tokens are then passed through the pre-trained BERT model, which generates a contextualized representation for each token in the input sequence, including the [MASK] tokens. The model is then trained to predict the original tokens masked out based on the contextualized representations generated by the model. The model generates a prediction for the masked tokens by applying a linear transformation (via a fully connected layer) to the corresponding token representation. The training loss is calculated as the cross-entropy loss between the predicted and original tokens. The objective of the MLM task is to minimize this loss and train the BERT model to generate high-quality contextualized representations for each token in the input sequence.

The NSP task is a pre-training task used to learn the relationship between pairs of sentences. The NSP task aims to train the language model to predict whether a given sentence follows another sentence in a text sequence. During NSP training, pairs of consecutive sentences are fed as input sequences to the model. Each sentence pair is labeled as a *next sentence* pair or a *random sentence* pair. In the case of a *next sentence* pair, the second sentence directly follows the first sentence in the text sequence. In the case of a *random sentence* pair, the second sentence is chosen randomly from the text sequence. The BERT model is then trained to predict whether a given sentence pair is a *next sentence* pair or a *random sentence* pair. A [CLS] token is added as a starting token and the sentence pair are separated with a [SEP] token. The model's output representation corresponding to the [CLS] token is then used for the NSP task. A fully connected layer takes this representation as input and performs binary classification to predict whether the next sentence follows the context sentence. The model is trained to minimize the binary cross-entropy loss between predicted and actual labels. The NSP

task is crucial because it helps the BERT model learn the relationship between sentence pairs, a key component of many natural language processing tasks.

After pre-training, the BERT model can be fine-tuned for specific NLP tasks. During fine-tuning, the pre-trained BERT model is combined with a task-specific output layer and trained on a labeled dataset for the specific task. One of the critical features of BERT is its ability to generate high-quality contextualized word embeddings, which capture the meaning of words based on their surrounding context.

2.5 Related Work

2.5.1 BM25

BM25 (Best Match 25), further referred to as BM25 is a ranking algorithm used in information retrieval for ranking documents based on their relevance to a given query and is an improved version of TF-IDF. It calculates a score given a query and document using the following formula:

$$TF\_IDF(D,Q) = \sum_{i=1}^{n} IDF\left(q_i\right) * \frac{f(q_i, D) * (k_1 + 1)}{f(q_i, D) + k * \left(1 - b + b * \frac{|D|}{avgdl}\right)}$$

where $f(q_i, D)$ is the frequency of the query term q_i in the document D, $|D|$ is the length of the document D, $avgdl$ is the average document length in the corpus. Both k and b are tuning parameters. $IDF(q_i)$ is the inverse document frequency of the query term q_i.

BM25 is designed to balance precision and recall by considering the frequency of the query terms in the document, the document length, and the frequency of the term in the entire corpus. The formula uses a term frequency normalization factor (k) and a document length normalization factor (b) to prevent over-representing long documents and highly frequent terms. The IDF factor down weights highly frequent terms and up weights rare terms, which can be more informative for the retrieval task.

Let's consider a simple example to illustrate how BM25 works. Suppose we have a collection of three documents:

- The quick brown fox jumps over the lazy dog.
- The quick brown dog is in love.
- The lazy dog is quick.

Suppose we want to rank these documents based on their relevance to the query *quick brown fox*. Let's calculate the IDF.

$$IDF(\text{"}quick\text{"}) = log(3/3) = 0$$

$$IDF(\text{"}brown\text{"}) = log(3/2) = 0.1761$$

$$IDF(\text{"}fox\text{"}) = log(3/1) = 0.4771$$

Next, we calculate the term frequency of each query term in each document:

- Document 1: f("quick") = 1, f("brown") = 1, f("fox") = 1
- Document 2: f("quick") = 1, f("brown") = 1, f("fox") = 0
- Document 3: f("quick") = 1, f("brown") = 0, f("fox") = 0

I choose to use values that have turned out to work pretty well for most corpora, setting $k = 1.2$ and $b = 0.75$ [9]. The average document length in the corpus, $avgdl = 7$. We can calculate the relevance score for each document using the BM25 formula. Let's first look at the score for Document 1.

$$Score(Document1, \text{"}quick\text{"}) = 0 * \frac{(1 * (1.2 + 1))}{(1 + 1.2 * (1 - 0.75 + 0.75 * (9/7)))} = 0$$

$$Score(Document1, \text{"}brown\text{"}) = 0.1761 * \frac{(1 * (1.2 + 1))}{(1 + 1.2 * (1 - 0.75 + 0.75 * (9/7)))} = 0.16$$

$$Score(Document1, \text{"}fox\text{"}) = 0.4771 * \frac{(1 * (1.2 + 1))}{(1 + 1.2 * (1 - 0.75 + 0.75 * (9/7)))} = 0.43$$

$$Score(Document1, \text{"}quickbrownfox\text{"}) = 0 + 0.16 + 0.43 = 0.59$$

If we do the same for documents 2 and 3, we will get the following scores:

$$Score(Document2, \text{"}quickbrownfox\text{"}) = 0 + 0.18 + 0 = 0.18$$

$$Score(Document3, "quick brown fox") = 0 + 0 + 0 = 0$$

Therefore, according to the BM25 score, Document 1 is the most relevant to the query *quick brown fox*, followed by Document 2 and then Document 3. This shows how BM25 can be used to rank documents based on their relevance to a query, even when there are differences in the documents' contents and lengths.

One of the advantages of BM25 is that it is easy to implement and can be applied to large-scale collections efficiently. It also provides good ranking quality for a wide range of retrieval tasks. However, BM25 has several limitations, such as the difficulty of selecting the optimal hyperparameters k and b for a given collection and the lack of support for modeling phrases.

2.5.2 TF-IDF with Logistic Regression

Combining TF-IDF with logistic regression to train a binary classifier is a practical approach that can be used for sentiment analysis on textual data. Previous research experiments using Twitter data to classify the text's opinion on news and scope the task to classify whether a text is *happy* or *unhappy* oriented. Their results show that a logistic regression classifier with SGD optimization and TF-IDF produces the most optimal result [2, Yousaf et al.].

TF-IDF, combined with logistic regression, has also been tested on the classification of short texts. Researchers have compared and evaluated different feature selection methods like TF-IDF, word2vec, and paragraph2vec in combination with classifiers like Naive Bayes, Logistic Regression, and Decision Trees. They conclude that combining TF-IDF and logistic regression with TF-IDF is among the methods that attain the highest accuracy [35, Wang et al.].

TF-IDF is a numerical method for measuring the importance of a word in a document corpus and is described in section 2.2.4. As described in subsection 2.1.7, logistic regression is a model used for binary classification tasks that predicts the probability of the positive class as a logistic function of a linear combination of input features. TF-IDF with logistic regression used in binary prediction tasks works as follows: The data needed to train this model would be raw text and a binary label. The TF-IDF technique is then applied to convert the raw text into a numerical representation that captures the importance of each word in the document. Once the text has been transformed into a numerical

	Word 1	...	Word M
Document 1	0.2	0.01	0
...	0	0.11	0.04
Document N	0	0	0.38

Table 2.4: TF-IDF Matrix showing documents along with TF-IDF values. Each row in the matrix represents a document, whereas a column represents a word. Make a note of the matrix sparsity that is because not all words are represented in all documents.

representation using TF-IDF, the logistic regression algorithm trains a binary classifier that can predict the binary label. Each row in the matrix corresponds to a document in the corpus, and the columns correspond to the words in the vocabulary. Each element in the matrix will therefore represent a TF-IDF value of a word in a document. The matrix is typically extensive and sparse since most documents only contain a small subset of the words in the vocabulary. We now use the output of the TF-IDF model as training data for the logistic regression model.

During training, the model learns to identify relevant patterns and features in the TF-IDF vectors associated with the binary labels. This learning process involves adjusting the weights and biases of the model's parameters using a loss function and the SGD optimization algorithm.

2.5.3 Passage re-ranking with BERT

Pre-trained BERT models have previously been shown to help develop passage re-ranking models [29]. This research looks at the task as a question-answering task where the user inputs a query and wants the passages returned ordered by the most relevant to the user's need. The algorithm is divided into steps. The first step involves using a standard mechanism like BM25 to obtain an initial set of passages. This set may contain thousands of potentially relevant passages from a more significant hub of passages. The next step is calculating each passage's score using the proposed model. It is then possible to do a re-ranking based on these passage scores. The final step is to pick the top ten or twenty passages with the highest score.

Their proposed method uses a pre-trained BERT model and adds a task-specific feed-forward layer with a single neuron as output. Keeping in mind that the [CLS] token holds the contextualized representation of the whole input, the feed-forward layer is added to this representation. The output of the model score s_i is an estimation of how relevant the

passage d_i is to a query q. They treat the query as Sentence A and the passage text as Sentence B. They are using truncation techniques if either query or passages exceed the limits. Notice that since we are concatenating the query and passage before feeding it to the model, the final ranking output score includes all term pair interactions between the query and document through all the transformer layers, making it an interaction-based model.

The model uses the pre-trained BERT_{LARGE} and is fine-tuned for passage re-ranking using the cross-entropy loss. They train their model on two benchmark datasets, MS MARCO, and TREC-CAR, and compare their results with existing state-of-the-art models. Their approach outperforms all ranking models by significant margins on two performance measures, showing that the BERT model can be used as a passage re-ranker. This interaction-based approach is recommended as a way to do fine-tune of the BERT model [12].

2.5.4 Representation Based Ranking with BERT

In contrast to the approach mentioned in 2.5.3, other representation-based approaches have been tested [36]. This research explores multiple ways of using the BERT model for document ranking. This approach uses the [CLS] token representation whose embeddings are treated as a representation of the input text. It is thus possible to input a query q and a document d in separate models, whereas the ranking score can be applied to both [CLS] tokens. This approach is representation-based since the query and document are fed individually without interaction. They research if it is possible to use a similarity measure like the cosine similarity function as a ranking score. The results are thus not comparable to the recommended interaction-based approach for fine-tuning BERT. They conclude that the model's performance is nearly random and highlight that BERT is an interaction-based matching model.

2.5.5 Tools for Systematic Review

There are multiple ways of finding relevant studies when creating a Systematic Review for a particular medical intervention. One way is to use search engines like Google Scholar. The researcher uses this tool by typing in a query and browsing the ranked results of documents. The browsing can be done by iteratively evaluating the title and abstract

of the study and then considering whether this study contains information related to the particular medical intervention. The best case for the researcher would be that all the relevant articles would be listed in the top results, avoiding him using unnecessary time to evaluate non-relevant studies. Unfortunately, this is not the case, mainly due to how Google Scholar is optimized. The objective of Google Scholar is to evaluate the importance of academic documents using researchers' criteria. It considers various features, such as the document's text, features from the author, the publication source, and the frequency and recentness of citations in other scholarly literature [30]. These features are gathered by *robots* or *crawlers* that run on the internet [31]. The score from each feature, as well as a weight specifying the importance of the feature, determines the search engine ranking results. It would be interesting to evaluate the search engines' exact ranking algorithm but unfortunately, Google has never released this information [10, p. 2]. Google Scholar is thus, in addition to considering the document text itself, also considering other features resulting in a ranking optimized towards different user's needs. Another limitation is the search engines' lack of reproducing the same result over some time. Studies have shown that Google Scholar did not satisfy reproducibility in reporting identical results for repeated identical queries [19, p.28]. Due to this and other factors, Google Scholar is considered a supplementary tool for finding relevant studies for systematic review [19, p. 16].

PubMed and Cochrane are considered principal tools for finding considered for finding relevant studies. [19, p.30]. With a series of filter options available, these search engines provide a highly organized database that is particularly advantageous for executing structured searches in medicine. Common filters include publication date, study type, article type, language, species, and age group. Using different boolean operators, they also allow users to create custom filters based on specific search criteria. Although, this functionality may be great if the researcher knows the specific search criteria of the study he is looking for. Usually, the researcher doesn't know these and only relies on a single search query.

Figure 2.16 shows the result from the Cochrane search engine when performing the search query in Figure 2.15. As you can see, 42 studies satisfy the defined criteria in the search. Using this search approach can enhance the precision of the results, but it may also reduce sensitivity. Consequently, the user may be unable to locate all relevant studies for a given medical intervention due to researchers' terminology and writing style variations. This approach may not be ideal for finding relevant studies on medical interventions unless the researcher is aware of the specific terminology used in all related studies, which is highly unlikely. While utilizing the Cochrane search engine does ensure the retrieval of

health-related studies, there is no guarantee that the top result will be the ones containing information needed in the Systematic Review.

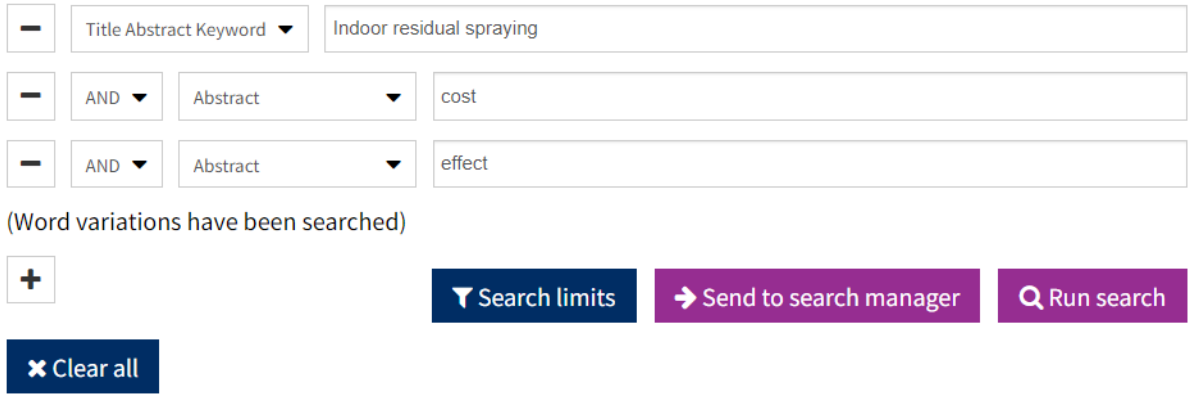

Figure 2.15: Example of a search query using Cochrane search engine. Three search criteria are defined and combined using the boolean AND operator. The first specifies a term that must be included in the study's title, and the other two specific terms must be present in the abstract section. Since the AND operator is used between them, the retrieved results of studies must satisfy all rows defined.

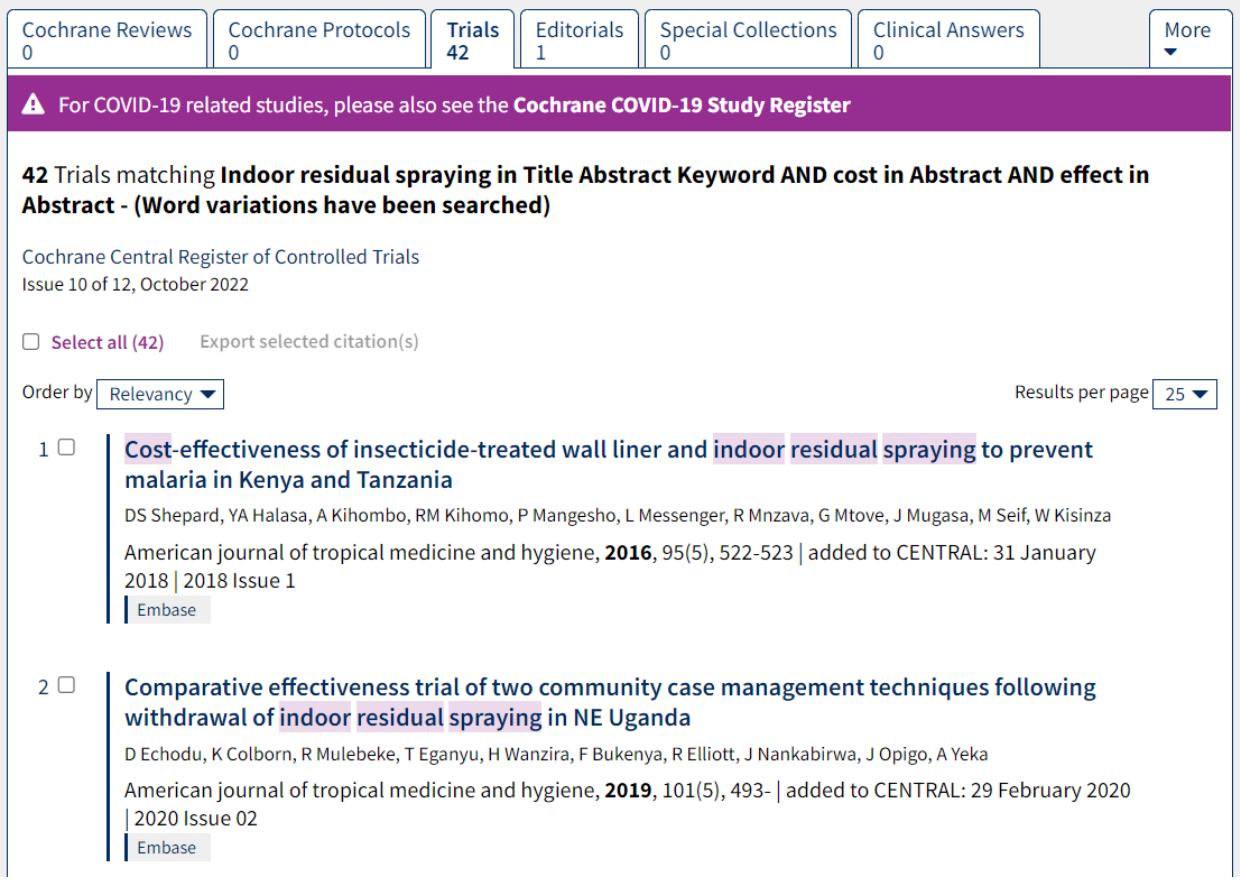

Figure 2.16: Figure showing a Cochrane search result that satisfies a filter criterion defined by a user. The highlighted words in purple are defined by the user to be included in the title of all returned results.

Covidence is another screening tool and the primary tool when authors from Cochrane are conducting systematic review[8]. The tool is a web-based software platform designed to help users with tasks like abstracts and full-text screening, conducting risk of bias, and performing data extraction. It works by first importing a series of citations (or studies) and then continuing with the title and abstract screening. The tool also facilitates the data extraction phase to become more manageable and seamless by providing a marking tool and inviting multiple users. The tool is thus great for managing the title and abstract screening, but it does not say which studies should be included. It depends on which studies the user upload to the tool and doesn't implement ML techniques for intelligence.

Rayyan.ai [21] is similar to Covidence by providing a web-based platform for uploading a list of studies and having humans in the loop. The tool differs by having multiple ML functionality, making it easier to screen titles and abstract. For example, you can detect duplicate studies, where each detected duplicate returns a percentage of how sure it is. It is up to the user to either trash or keep the suggested duplicate. Also, the tool facilitates

an ML functionality for determining which studies to be included and which not. This is done by first making the user screen all studies and mark them as relevant, non-relevant, or undecided. Then, the tool extracts textual and meta-data features for each relevant and non-relevant document. This feature representation of a document is then used as input to a random forest classifier to predict accurately included studies with high recall [18]. Then the human label is used for training resulting in a binary prediction problem. Then, for each study labeled as undecided, the tool can calculate the probability of the study being included based on how the labeled data trained the model [1, p. 1-2].

Chapter 3

Data

This chapter introduces and discusses the data used in the present thesis, including the data source from which it was gathered. The chapter highlights the processing steps required to arrange the data in the desired format. Furthermore, the chapter concludes by offering an analysis of the data, which underscores the data's characteristics and potential limitations.

The code implementation can be found in the following GitHub repository: Master Thesis.

3.1 Systematic Review Data

Systematic reviews are an essential tool for healthcare professionals and policymakers to make informed decisions about the effectiveness of different interventions. Cochrane, a global independent network of researchers, professionals, patients, and carers, has produced high-quality systematic reviews [16].

The systematic review data from Cochrane's are valuable for researchers to gain valuable insights into the latest research findings on a wide range of health topics, ranging from the efficacy of different medications to the effectiveness of particular medical interventions. This dataset includes approx. 10 000 Systematic Reviews, each including the systematic review's title and the list of related and non-related studies. A systematic review contains a comprehensive overview of for a particular research question, the methods used to address it, and the main findings. The dataset also includes other information, but only the title and the list of related and non-related studies are interesting for my task.

3.1.1 Dataset

The dataset obtained from Cochrane comes with some challenges. One is that the related and non-related studies are only given by the study's title, year, and list of authors. Whereas what we need is the abstract of the study. The dataset requires further processing to access the abstract of each study.

PubMed is a free online database of biomedical literature and citations. It provides access to over 35 million citations from MEDLINE, life science journals, and online books. PubMed provides access to many medical studies and other literature, making it an important tool for researchers, clinicians, and healthcare professionals [27]. In addition, they have developed a Python library, making it possible for a Python program to search for studies.

Thus, obtaining a particular study's abstract is possible using the PubMed Python library. Hopefully, the abstract will provide additional information better to understand the relationships between the systematic review title and gain a deeper understanding of the study presented. Later in this chapter, you will see how to retrieve a study's abstract in PubMed using a string-based query from the Cochrane data.

This thesis focuses on document ranking models that are trained using the so-called pointwise learning-to-rank method. See Chapter 2, section 2.2.5 for more details on pointwise training. Using the pointwise method for study ranking, a binary classification model can assign a binary score (e.g., 1 or 0) to each study based on its relevance to the query. This model can be trained on a dataset of studys and their corresponding relevance labels.

The processing algorithm works as follows. We start by iterating over all systematic reviews. For each systematic review, we want to iterate over all associated studies. Keep in mind that an associated study can be either related or non-related. We now want to determine if we can find the correct study in PubMed. We then use the title of the study as an input query towards the PubMed library. The PubMed library allows specifying how many search results we want in return. Based on the returned studies, we need to determine which study is the correct reference in the Systematic Review data. It is important that the correct study returned from PubMed results are the actual study listed in the systematic review. If not, then the dataset will be misleading and useless. Keep in mind that it is not guaranteed that the study exists in the PubMed database, and thus, I cannot just pick the top result. Also, keep in mind that there may exist

studies with very similar titles to the one I am looking for. Also, remember that in the returned results from PubMed, the correct study may not be listed as the first. All these factors are important when developing a data creation algorithm.

Table 3.1 highlights some titles from the PubMed data that are close to similar to the Systematic Review data. The yellow color coding show that there are minor differences. These differences may indicate two different studies or inconsistency in how the data sources have included the same study in their database. Thus, further analysis of the study's author and year is required to determine if the studies are the same.

Title (from Systematic Review)	**Title (from PubMed)**
The effects of abdominal decompression on pregnancy complicated by the small-for-dates fetus	Letter: The effects of abdominal decompression on pregnancy complicated by the small for dates fetus.
[Repair of episiotomies with synthetic suture material]. [Bulgarian]	[Repair of episiotomies with synthetic suture material].
Failure of naloxone to modify the anti-tobacco effect of acupuncture	[Failure of naloxone to modify the anti-tobacco effect of acupuncture (author's transl)].
Impact of home patient telemonitoring on use of beta-blockers in congestive heart failure	Impact of home patient telemonitoring on use of β-blockers in congestive heart failure.
Group discussions with parents have long-term positive benefits on the management of asthma with good cost-benefit	Group discussions with parents have long-term positive effects on the management of asthma with good cost-benefit.

Table 3.1: Samples of document titles that are almost similar in the Systematic Review and PubMed data. The yellow coding highlights the differences in the titles and tells that the differences are minor. For these type of cases, other features have to be considered to determine if the article from PubMed is the same as the one in the Systematic Review.

Further analysis needs to be considered for cases where the title is not similar enough. Let's look at the similarity between the authors listed in the Systematic Review and PubMed.

Authors (from Systematic Review)	Authors (from PubMed)
Varma TR, Curzen P	Curzen P
Nikolov A, Dimitrov A, Iliev D, Krsteva K	Nikolov A, Dimitrov A, Iliev D, Krŭsteva K
Boureau F, Willer JC	Boureau F, Willer J C
Antonicelli R, Mazzanti I, Abbatecola AM, Parati G	Antonicelli Roberto, Mazzanti Ilaria, Abbatecola Angela M, Parati Gianfranco
Hederos CA, Janson S, Hedlin G	Hederos C-A, Janson S, Hedlin G

Table 3.2: Samples of documents where the authors that are almost similar in the Systematic Review and PubMed data. The yellow coding highlights the differences. The table shows that the authors of some studies are written differently.

Table 3.2 highlights how authors for a particular study are listed in the Systematic Review Data and PubMed data. The rows in the Table are authors from the same documents as in Table 3.1, listed in the same order. The yellow color shows the differences where differences may be spacings, dashes, special characters, and if the last name is written all out.

Based on the mentioned considerations, I developed a precision-related generation algorithm to create the dataset. By precision-related, I mean that it is better to include the datapoint if I am close to 100% sure that it is correct. This may cause a risk of not including all potentially correct studies with the tradeoff of being quite sure that the ones collected are correct. Comparable data features for a particular study included in the Systematic Review, and PubMed are the title, year, and authors. The most optimal way of choosing the correct study would be to compare all these three features and see if they are all equal. Unfortunately, this is not possible for multiple reasons. First, it varies if the year of publication is included in the PubMed study. Second, it varies how the authors are listed. Sometimes, all authors are listed with both first and last names. Sometimes, all authors are listed by only their last names. Regarding the title, this should be identical unless stored differently in the different database systems.

I have therefore chosen to use both the author and title when considering finding the correct study. And the similarity criteria are as follows: Calculate the pattern similarity between the titles using Gestalt Pattern Matching, see 2.1.11 for more details. If the similarity is above 0.98 (98% in similarity), the PubMed study is the right one. Otherwise, if the similarity is above 0.85, the study may be the right one, and we need to look at

the authors and year. If the year is the same and the author similarity is above 0.75, the PubMed study is the right one.

The final step is to make the label. If the study is relevant, the label is equal to 1. If non-relevant, the label is equal to 0. To make the dataset even more fine-grained, only queries with a minimum number of 30 associated documents were included. This was with the intention of having multiple relevant and non-relevant documents making it possible for models to learn general patterns across many documents per query.

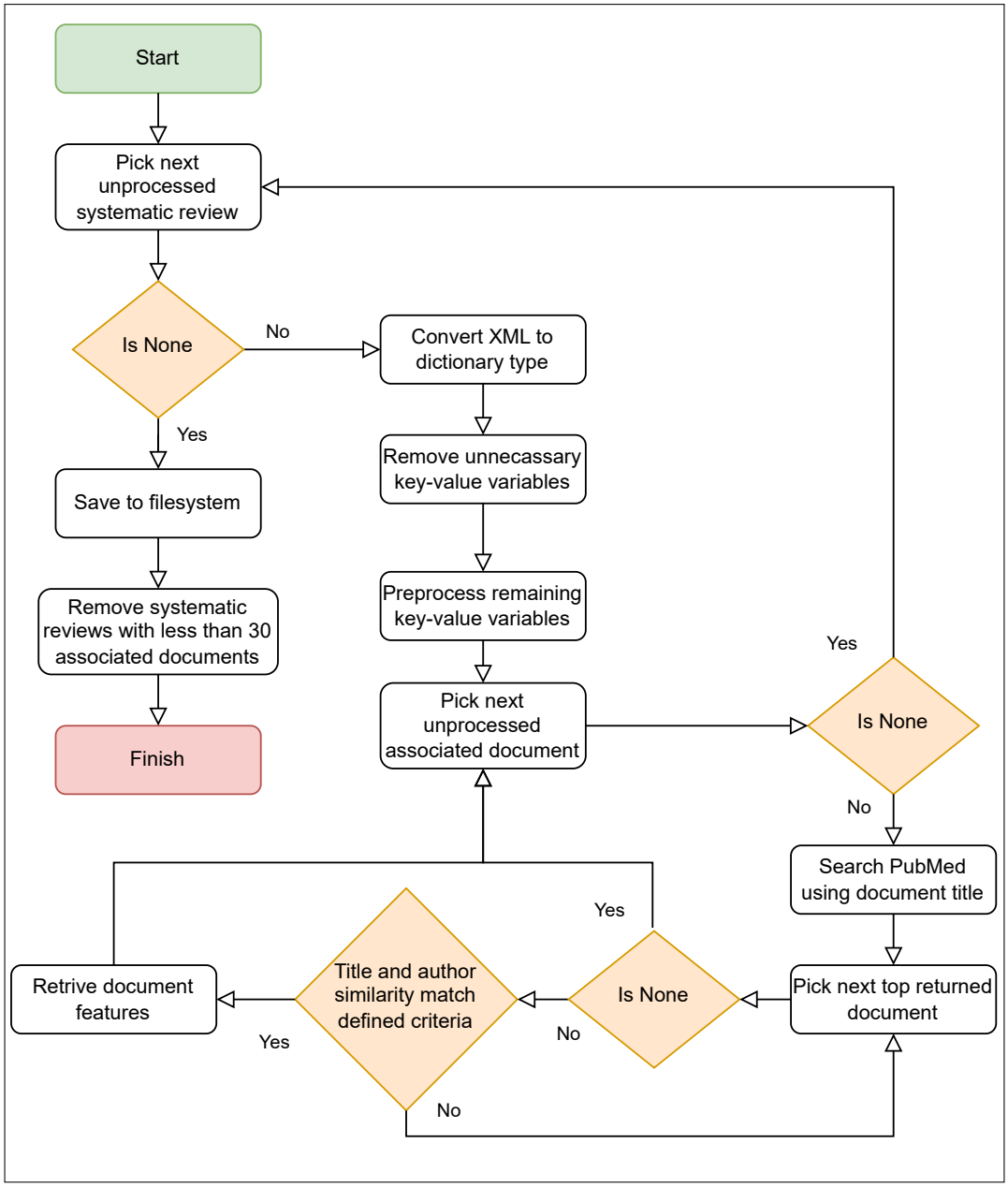

Figure 3.1: Flowshart showing how the dataset is created. In short, the script is first iterating over all systematic reviews. Next, we iterate over all relevant and non-relevant documents for the particular systematic review. Then, we search PubMed for each document, trying to identify the correct document. If a returned document from PubMed satisfies similarity criteria, we retrieve the document features and analyze the next document.

We now have all we need to create a data point: the title of the systematic review(**Query**), the title of the study(**Document Title**), the abstract (**Document Abstract**), and **label**. We will further refer to these three features as our text features. Some BERT-based models are dependent on passing in an extra-relevant abstract as in-

put to the model. Therefore, an extra text feature is created which is basically just a random sample of a true labeled document abstract for the particular query. Table 3.3 shows how the final dataset looks, where each row corresponds to a data point.

Query	**Document Title**	**Document Abstract**	**Label**
Antibiotics for treatment of sore throat in children and adults	Do patients with sore throat benefit from penicillin? A randomised double-blind placebo-controlled clinical trial with penicillin V in general practice	The effect of antibiotic therapy in sore throat is questionable and this dilemma has been complicated by the emergence of multiple resistant strains of micro-organisms...	1
Antibiotics for treatment of sore throat in children and adults	Aetiology of acute pharyngitis and clinical response to empirical therapy with erythromycin versus amoxicillin	One hundred and eighty-nine adults with acute pharyngitis had culture and serological evaluation for group A beta haemolytic streptococci (GABHS), Mycoplasma pneumoniae, and Branhamella catarrhalis...	0

Table 3.3: Pre-processed systematic review data with three text features and one label. The table includes two data points, one positive and one negative sample. It shows that for a particular systematic review title, we have two different studies associated with it where one of them is related, and the other is not. The data points in this table are randomly collected data points.

An alternative to this automatic data creation algorithm is to manually iterate through the Systematic Review data, evaluate the title, year, and authors, and then ensure the correct study is selected. However, this would take an extremely long time and is not within the scope of this thesis.

3.1.2 Exploratory Data Analysis

By analyzing these metrics, we can gain a deeper understanding of the characteristics of the text data and how they relate to the research questions you are investigating. Since, the BERT model has a maximum sequence length of 512 tokens, which restricts the length of input sequences, it is interesting to determining the word counts for each text feature. The analysis therefore compute, after applying whitespace tokenization, the average number of words in each text feature. In addition, it is preferred with an analysis of the distribution as plots of histograms. Altough, counting words using whitespace tokenization is only relevant for methods using this tokenization approach. For the methods using BERT based approaches we need to do the same analysis but tokenizing the text using WordPiece tokenization. Since this method divide words into subwords, the number of tokens is likely to increase.

3.1.3 Exploratory Data Analysis Results

I will now delve into the results from the exploratory data analysis, with a specific focus on the word and token counts in the different text features.

The dataset comprised a total of 94 823 data points. After applying a split ratio of 85, 7.5 and 7.5 for the train, val and test set respectively, the total points were 80268, 7633, and 6922, respectively. The further analysis then only analyzed the training data. Whether this quantity of datapoints is sufficient for training a model for the task at hand remains uncertain.

The Figure 3.2 shows a histogram of word count for the queries using whitespace tokenization. The results show that the most of the queries falls within the 5-15 word count range, with a mean of 10 words per query. This observation suggests that the query uses a relatively small portion of the input tokens to the BERT model to accommodate longer document tokens while preserving meaningful information.

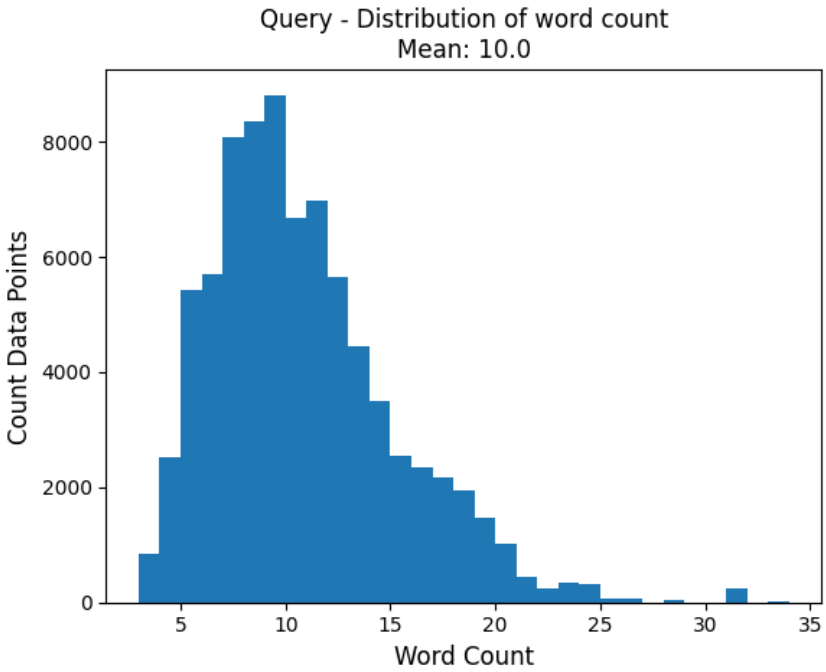

Figure 3.2: Histogram of word counts in systematic review titles. The y-axis displays the number of documents, and the x-axis shows the range of word counts. The mean of words per title is 10.

The word count distribution of the document titles in the dataset exhibits similarity with that of the queries, with a mean of 16 words per title. Thus, including the document title feature in Sentence 2 when training a BERT model won't use up a lot of input tokens to the BERT model and may provide a greater ranking.

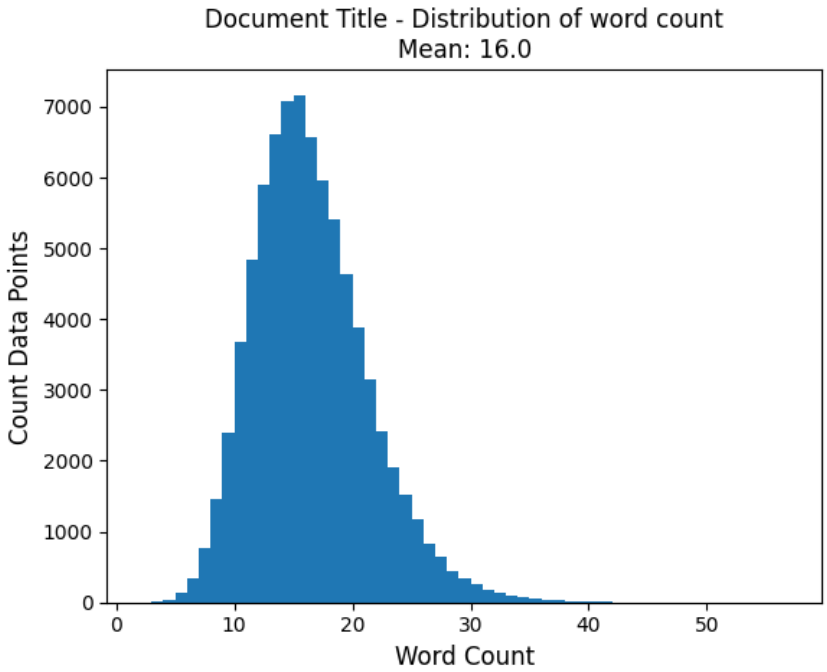

Figure 3.3: Histogram of word count in document titles. The y-axis displays the number of titles, and the x-axis shows the range of word counts. The mean of words per title is 16.

In contrast to the small mean of word counts in the queries and document titles, the histograms of token length for document abstracts show a distribution with a mean of 230 words per abstract.

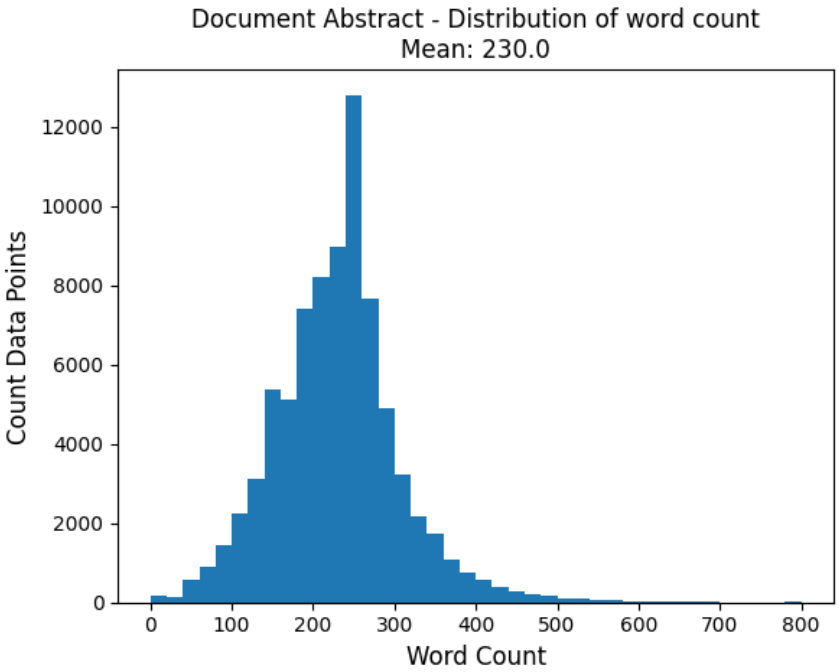

Figure 3.4: Histogram of word count length in the document abstracts. The y-axis displays the number of abstracts, and the x-axis shows the range of word counts. The mean of words per title is 230.

Figure 3.5 shows a histogram of the count of the BERT tokenized concatenation of query and document abstract. The figure demonstrates a clear increase in the length with a mean of 359. Both because we are concatenating two text features and the use of a subword tokenizer by BERT. These findings suggest that the subword tokenization process significantly increases the length of the input sequences, which can impact the ability to process and analyze the data using the BERT model. The percentage of datapoints that is truncated is 12%. Furthermore, when combining the text features query, document title, and document abstract, an even greater proportion of the dataset exceeds the 512-token limit. Specifically, 16% of the data points will be truncated. Figure 3.6 shows this distribution.

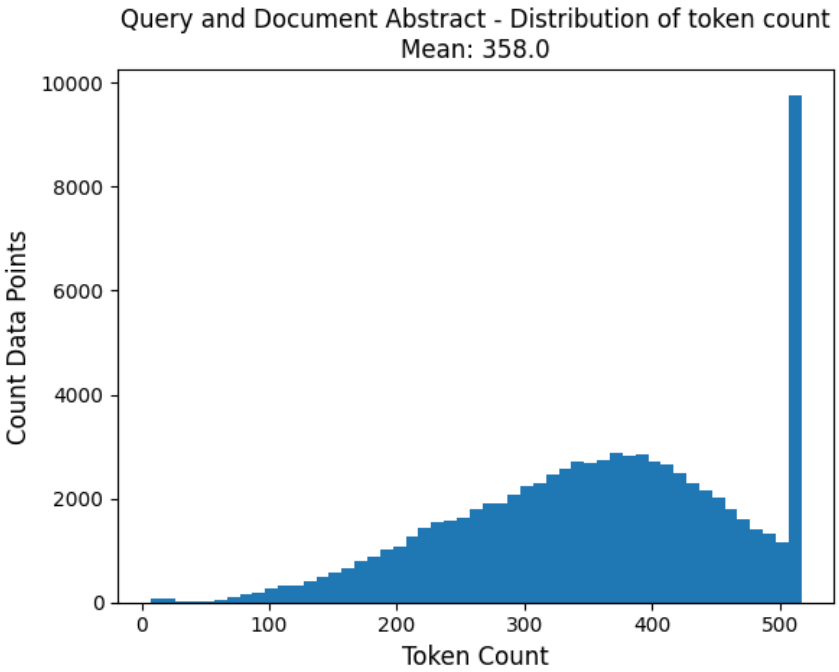

Figure 3.5: This histogram shows the distribution of token counts when combining the query and document abstract. It is the count of tokens after tokenization with the BERT-tiny tokenizer has been applied. The y-axis displays the number of data points, while the x-axis shows the number of token counts. The histogram shows a skewed distribution, with a median of 359 tokens. Notably, a significant proportion of the data points hits the 512-token limit, indicating that many documents have been truncated during processing.

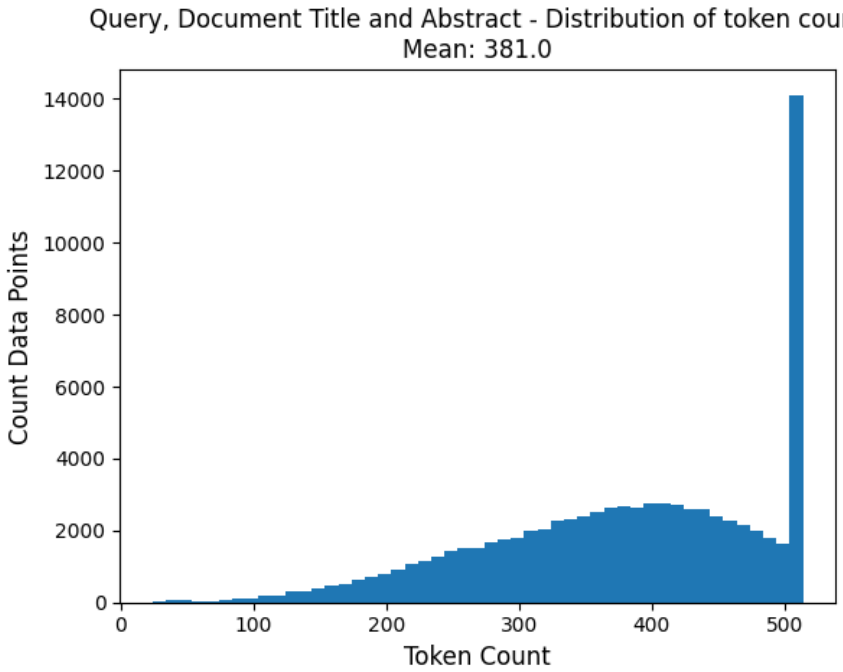

Figure 3.6: This histogram shows the distribution of token count in the combination of query, document title, and document abstract after tokenization with the BERT-base tokenizer. The y-axis displays the number of abstracts, while the x-axis shows the range of token counts. The histogram shows a skewed distribution, with a median of 380 tokens. Notably, a significant proportion of the data points hits the 512-token limit, indicating that many documents have been truncated during processing.

These findings highlight the importance of considering the impact of subword tokenization on the length of input sequences, particularly when using the BERT model for text analysis. Effective preprocessing and handling of input sequences are critical to ensure that important information is not lost during the modeling process. Possible strategies to handle longer input sequences include cleaning the data by removing stopwords to reduce the number of tokens or truncating the text feature before tokenizing it to avoid truncation in the middle of subword tokens. Another possibility is splitting the data and inputting it into individual BERT models. However, using individual models may sacrifice BERT's interaction characteristics, as multiple text features are not inputted into the same model.

Figure 3.7 shows a slight predominance towards the False labels. This makes regarding that there are usually a lot more non-relevant documents than relevant documents when creating the systematic review. Although it is optimal with a close to equal distribution of the binary labels, this distribution is totally fine to train a neural network. The distribution is 65% to 35% in favor of the non-relevant documents.

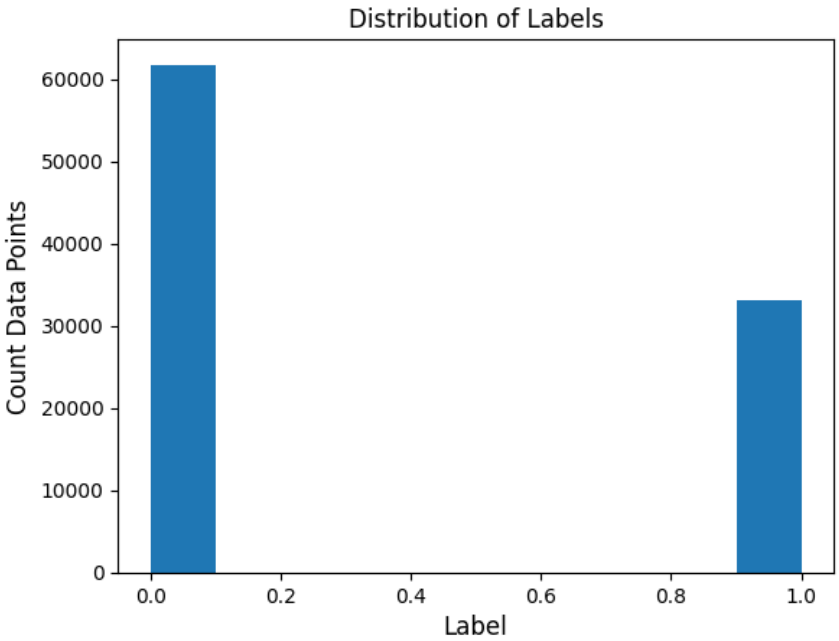

Figure 3.7: Histogram showing the number of True and False labels. The y-axis shows the number of datapoints whereas the x-axis shows the label, ranging from 0 to 1. The histogram shows a slight predominance towards the False labels with a distribution of 65% to 35%.

Chapter 4

Method

The method chapter describes the various approaches implemented for document ranking, utilizing binary classification as the underlying methodology. The research aims to improve the MAP of document ranking for Systematic Review data by employing different techniques, where we consider reducing a loss function equivalent to increasing the MAP score. The first two approaches, Pubmed Ranker and BM25, serve as baseline models for comparison. Then I describe the first approach, namely TF-IDF with logistic regression. Additionally, a series of approaches are explored that leverage the power of pre-trained BERT models. These approaches utilize different strategies and variations of BERT models to enhance the ranking performance. I will refer to them as Interaction-BERT, Representation-BERT, Dual-Interaction-BERT, and Triple-Representation-BERT.

It is important to note that several elements of the methodology were held constant across all our methods. These included the type of optimizer, train-test split, and loss function. A series of pre-trained BERT models are available. This thesis only examines the use of the Tiny-BERT [5], the minor type of BERT that uses only two encoders and has a hidden dimension size of 128.

Initially, the dataset was split into a train, val, and test set with proportions 85%, 7.5%, and 7.5% respectively. The size of the different sets was approx. 80 000 for the test set, 7 000 for the dev set, and 7 000 for the test set. The mean loss, accuracy, and MAP were calculated after each epoch.

The implementation utilizes the Python library PyTorch as a training framework for training neural networks. It also uses the transformer library that facilitates loading pre-trained language models.

The code implementation can be found in the following GitHub repository: Master Thesis.

4.1 Pre-Processing

The preprocessing section of this study plays a pivotal role in preparing the data for document ranking, considering the different methods utilized. The first method, which employs TF-IDF, involves training a TF-IDF model using a corpus of text documents. With the available data, namely the query, document title, and document abstract, multiple ways exist to combine these elements and create an interaction-based model.

In the case of using TF-IDF as an interaction-based model, one approach is to concatenate the query, document title, and document abstract into a single text document. This merging process allows the different textual components to interact, capturing potential relationships and semantic information within the document. By considering the document as a whole rather than separate entities, the TF-IDF model can better capture the context and relationships between the textual elements. This comprehensive representation enhances the analysis for document ranking. It enables more accurate word importance and relevance assessments within the document. The text for a single data point will therefore look like either of these:

$$QA = \text{query} + \text{document abstract}\,,$$

$$QTA = \text{query} + \text{document title} + \text{document abstract}\,,$$

where QA stands for Query-Abstract and QTA stands for Query-Title-Abstract. For the Interaction-BERT models, the preprocessing steps differ slightly. Remember that BERT was trained using the NSP where two sentences were concatenated with a [SEP] token. In the interaction-based models, we must place this [SEP] token in the preprocessing step. In an interaction-based model, where multiple texts are inputted into a single BERT model, the query is always considered Sentence 1. To form Sentence 2, two combinations of the remaining text features are explored. This includes using only the document abstract or concatenating of document title and abstract. The query and document are concatenated to tokenize the input using a [SEP] token. The text feature will therefore look like either of these:

$$QA = \text{query} + [SEP] + \text{document abstract}\,,$$

$$QTA = \text{query} + [SEP] + \text{document title} + \text{document abstract}\,,$$

where QA is read 'Query-Abstract' and QTA is read 'Query-Title-Abstract.' In contrast, for the Representation-BERT approach, each text feature is separately fed into the BERT model during preprocessing. This means the query, document title, and document abstract are treated as individual inputs to the BERT model, enabling separate representations for each feature. The text feature will therefore look like this:

$$x_i = \begin{cases} \text{query} \\ \text{document title} \\ \text{document abstract}\,. \end{cases}$$

Let's look at a model architecture we refer to as Dual-Interaction-BERT. The text features are now preprocessed differently for this architecture when creating the input data. The data will be processed in the following manner:

$$x_i = \begin{cases} \text{query} + [SEP] + \text{document abstract} \\ \text{query} + [SEP] + \text{relevant abstract}\,. \end{cases}$$

Finally, we have the triple representation-based BERT model, where the text must be handled differently. The main difference between this and Representation BERT is that we drop the document title and feed a relevant abstract instead. Therefore the data is preprocessed in the following way:

$$x_i = \begin{cases} \text{query} \\ \text{document abstract} \\ \text{relevant abstract}\,. \end{cases}$$

Overall, the preprocessing section encompasses the specific steps undertaken for each method. These preprocessing strategies are essential in preparing the data for subsequent analysis and ranking of documents.

Moving on to text cleaning, an optional step in the preprocessing phase, we have implemented one cleaning function that can be applied to the text features. Cleaning is applied before concatenating the texts to avoid messing with the [SEP] token. Each text feature can undergo no or complete cleaning and is set before training a model. This is

thus a hyperparameter to be set to either True or False. The cleaning method involves converting the text to lowercase, removing line breaks, eliminating links, removing stopwords, removing numbers, and performing lemmatization. The Python package nltk is used to retrieve a list of English stopwords. The package provides a list with a total of 179 stopwords, including *i*, *me*, *my*, and *myself*. Also, in the lemmatization step, a class from the nltk library is used, namely WordNetLemmatizer. This method aims to create a cleaner and more standardized representation of text features.

The text can be passed to the BERT tokenizer when it is in the desired format. This tokenizer adds the [CLS] token at the start and an [SEP] at the end in addition to tokenizing the whole text using the BERTs subword tokenization technique. Although it does not add the separator token since it doesn't know where the first sentence ends and the second sentence start. This is therefore added at an earlier stage.

4.2 PubMed Ranker

Since the PubMed Ranker is already trained, I don't need to consider preprocessing any text for training. The PubMed Ranker implemented in this study focuses on ordering documents retrived from PubMed. It submits each query to PubMed and evaluates the resulting document rankings. To interact with PubMed using Python, the pymed library is employed. Initially, the number of studies retrieved from PubMed is set to 1000, to make sure to fetch all possible relevant documents. The library will return a lower amount of documents if the search engine is not capable of finding the given amount. To ensure comparability with the systematic review, any documents published more recently than the publication date of the review are excluded. This ensures that the ranker aligns with the information available to the systematic review researchers. Since the documents obtained from PubMed are already ordered, this order is maintained and considered the final ranking. Furthermore, based on the analysis presented in Chapter 3 of the thesis, it was observed that the distribution of relevant and non-relevant documents is 65% towards 35%. Therefore, the top 65% of the documents are labeled as 1 (relevant), while the remaining documents are labeled as 0 (non-relevant). This labeling scheme proves helpful when calculating both MAP and accuracy. In cases where PubMed returns documents that cannot be found in our dataset, these documents are assumed to have a true label of 0 since they were available when the systematic review was conducted but was not included in the relevant or non-relevant document lists.

4.3 BM25

Implementing the BM25 ranker utilizes an external library called rank-bm25 instead of being developed from scratch. This library offers a convenient class named BM25Okapi, which simplifies the training and utilization of the BM25 model. The documentation from the library does not include which k and b parameters they use. The class requires a list of tokens as input, where each item represents a tokenized document. In this particular implementation, both the query and document abstract were combined, cleaned, and tokenized using whitespace tokenization and the same cleaning function as described initially in this chapter. It is important to note that this approach does not rely on a training model but instead applies statistics to a collection of documents. Therefore, the evaluation of its performance was solely based on its performance on the val set.

4.4 Optimizer

The same optimizer was used on all BERT-based and logistic regression-based models. During training, I used the ADAM optimizer with a linear schedule warmup, where the warmup was set to 20% of the training steps. Pytrch Adam optimizer was used.

4.5 TF-IDF with Logistic Regression

Moving on to TF-IDF with logistic regression, a systematic procedure is followed. This architecture is similar to related work 2.5.2 with the difference of treating the task as a ranking task instead of a classification task. After the splits, the data is cleaned individually if cleaning is chosen.

Next, the TF-IDF model is trained using the training data. This was done using a Python library sklearn where the class TfidfVectorizer was used. This facilitates training TFIDF models by just passing a list of texts. The trained TF-IDF model returns a TF-IDF matrix with dimensions n x m. Here, n represents the total number of documents in the training set, while m signifies the number of tokens. The hyperparameters chosen were a min_df = 8, a threshold for skipping terms with a lower frequency of the given value. We also set the ngram_range = (1,3), which specifies how many combinations of

terms we should create a TF-IDF value for. By setting ngram_range = (1,3), we create TF-IDF terms on two and three-word combinations given that they occur more than the min_df, threshold resulting in a richer TF-IDF vocabulary.

Following this, the logistic regression model is introduced. The TF-IDF matrix serves as the input for training the logistic regression model. A batch size of 16 is set for efficient training. Multiple models with different hyperparameters were trained to identify the best-performing model.

4.6 Interaction-BERT

This method is based on the approach presented in related work, 2.5.3. I use the query as Sentence 1 and experiment with different variants of the document features as sentence two. If the document is related, we have a training sample with label 1 indicating an is-next label. If the label is 0, the training sample is an is-not-next label.

The first step is data preprocessing, explained in detail in 4.1. The next step is initializing the model. The pre-trained BERT model is initialized with its pre-trained weights, and then a task-specific output layer is added on top of the model's $R_{[CLS]}$. The type of task-specific neural network, including the amount of feed-forward layers and activation function, depends on whether the BERT parameters are frozen or not, explained in more detail in Section 4.5. Since we have a binary classification task, the linear feedforward layer must have one output neuron, which is used to predict the binary classification output.

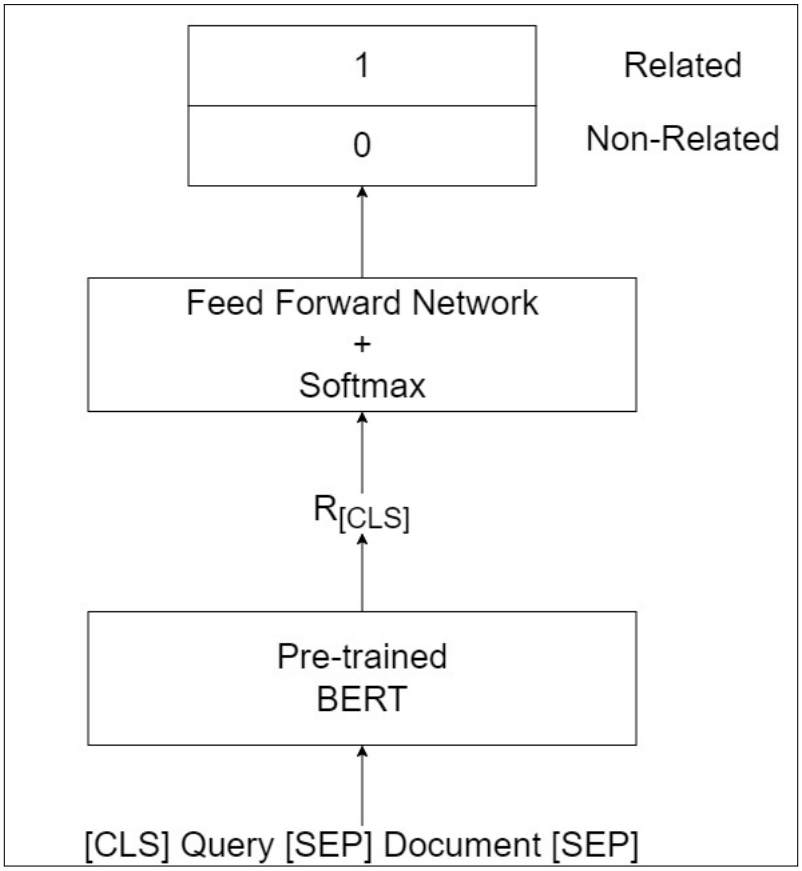

Figure 4.1: The model architecture combines a query concatenation and a document as input. The $R_{[CLS]}$ representation of the BERT model is then passed to a feedforward network, which generates a binary prediction of 0 or 1. The output will be used the rank all documents for a given query.

Figure 4.1 shows how the tokenized data is fed as input to the pre-trained BERT model. The model will return a contextualized representation for each input token, but we are only utilizing the representation of the [CLS] token on the last hidden layer. This is fed as input to the untrained feedforward neural network, which outputs a single numerical prediction for a particular data point.

Now that the customized BERT model is defined, it must be trained. It is now possible to only train the whole model, thus optimizing the parameters of the pre-trained BERT model and the newly initialized untrained neural network. Or we can freeze the parameters of the pre-trained BERT model and only train the untrained network added on top. The model is trained on the binary classification task data using backpropagation and gradient descent. The weights are updated to adapt to the task-specific data.

4.7 Representation-BERT

Another way to use pre-trained BERT-based models for binary classification is to use the model as a language representation tool to extract contextual information from text inputs. This method is similar to related work 2.5.4 with the difference of adding a neural network on top of the concatenation of the output representation instead of using comparison metrics, making it suitable for a ranking model. As mentioned in the explanation of BERT, 2.4.6, the [CLS] token holds the contextualized embedded representation of the whole input text. Instead of concatenating the query and document and feeding it into a single BERT model, we can feed each input to individual BERT models. The BERT models generate embeddings for each input sequence, which are then passed through the output layer of the model to extract the [CLS] token representation. This token captures the overall meaning of the input text and serves as a summary representation of the input.

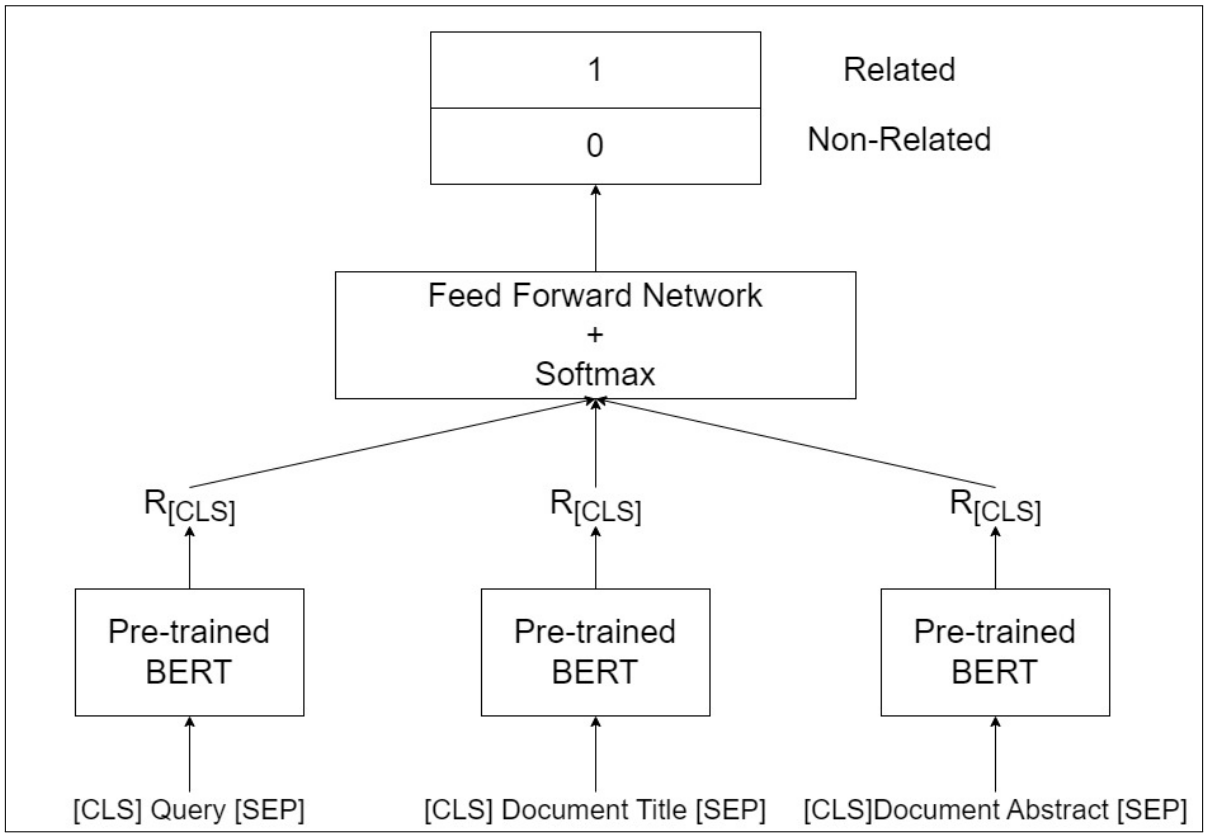

Figure 4.2: By leveraging individual BERT models and combining their $R_{[CLS]}$ tokens, a representation-based approach is achieved. The Figure shows three different text features that are passed into individual BERT models where the concatenated features flow through feedforward layers, culminating in a single neuron, predicting 0s and 1s.

Figure 4.2 illustrate how we can feed each text feature separately to obtain a contextualized representation. The [CLS] token representations extracted from each input sequence are combined into a single vector before being fed to the neural network component.

4.8 Dual-Interaction-BERT

Using two dual BERT models involves incorporating both the query and document information and a relevant document to enhance the document re-ranking process. This approach takes advantage of the unique insights provided by including a relevant document as an input feature.

The query and document are concatenated in the first BERT model using the [SEP] token as a separator. This concatenated input allows the model to capture the contextual information from both the query and the document simultaneously. The BERT model processes this concatenated input, and the $R_{[CLS]}$ token representation, which contains the aggregated information of the entire text sequence, is extracted. The query is concatenated with a relevant document in the second BERT model. This relevant document serves as an additional input feature, providing specific contextual information that is known to be relevant to the query. By including this relevant document, the model can benefit from the knowledge contained within it, potentially leading to improved understanding and ranking accuracy. Similarly, the BERT model processes this concatenated input, extracting the $R_{[CLS]}$ token representation.

The [CLS] token representations from the first and second dual Concat-BERT models are then concatenated. The concatenated representation is subsequently passed to the neural network component, which performs a binary classification task to predict a relevance score of 0 or 1.

Including a relevant document as input can be beneficial for several reasons. Firstly, the relevant document provides additional context to help the model better understand the query and the target document. It serves as a reference point, providing information about what constitutes relevance for a particular query. It allows the model to learn from the similarities and differences between the relevant document and other documents, intentionally leading to improved performance when ranking new, unseen documents.

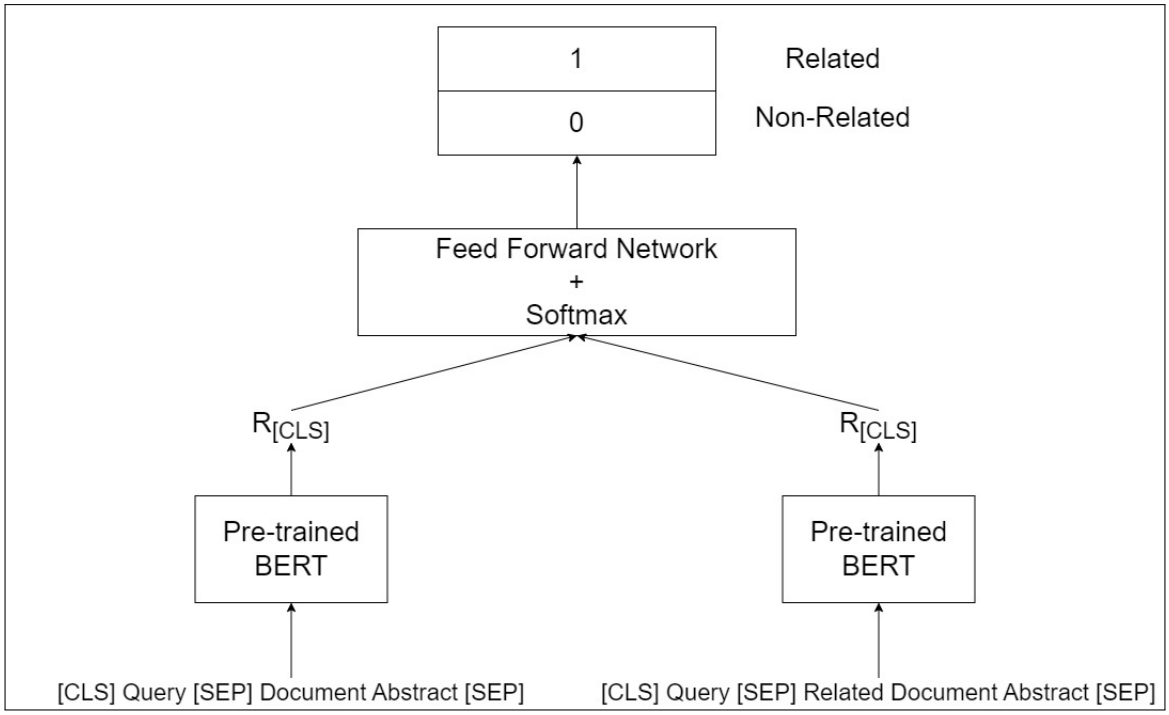

Figure 4.3: Illustration of a Dual BERT Model for Document Re-ranking. This architecture employs two interaction-based BERT models to enhance document ranking. The first BERT model receives a concatenated input of a query and a document, while the second receives a concatenated input of a query and a relevant document. The $R_{[CLS]}$ outputs from both models are concatenated and fed into a feedforward network, which generates a binary prediction of 0 or 1.

4.9 Triple-Representation-BERT

The Representation-Based BERT Model Ensemble is designed to leverage the power of BERT in capturing contextual information from multiple text features, namely the query, document, and relevant document. In this approach, the query, document, and relevant document are separately passed through their respective BERT models.

The $R_{[CLS]}$ token representation, which captures the aggregated information of the entire input sequence, is extracted from each BERT model. These representations are then concatenated to create a combined representation of the query, document, and relevant document. The concatenated representation is subsequently fed to the neural network component, which is responsible for making a binary prediction of 0 or 1.

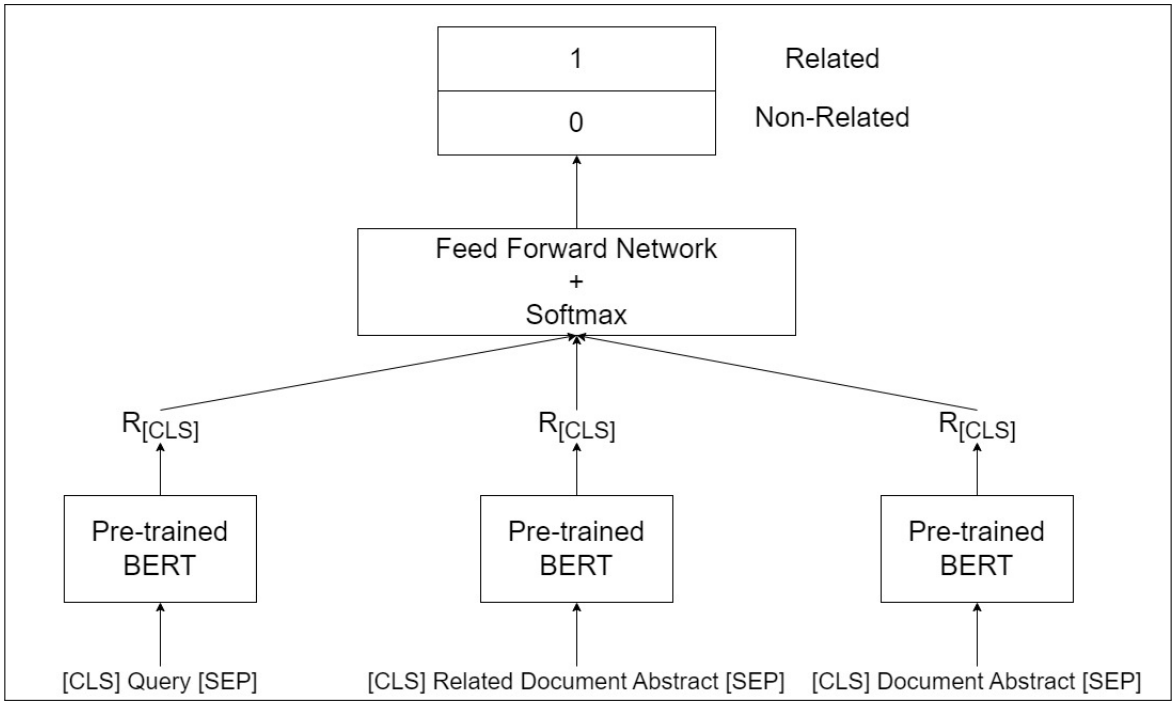

Figure 4.4: Illustration of a Triple Representation-Based BERT Model. This model architecture utilizes three individual BERT models, each dedicated to processing a different text feature: query, document abstract, and relevant document abstract. These text features are separately fed into their respective BERT models, and the $R_{[CLS]}$ token representation from each model is concatenated. This concatenated representation is then passed through a feedforward network, which generates a binary prediction of 0 or 1.

4.10 Feed Forward Network

As mentioned, all BERT-based models utilize a feed-forward network on top of the $R_{[CLS]}$ token or the concatenation of the $R_{[CLS]}$ tokens. The feed-forward network component is slightly different depending on whether the BERT parameters are frozen. By freezing, I mean freezing the parameters such that the parameters aren't updated throughout the training phase. Only the feedforward parameters are.

If the BERT parameters are not frozen, the model uses a dropout layer and a fully connected feed-forward layer with one single neuron as output. Contrarily, if the BERT parameters are frozen, the feed-forward network is a bit deeper. For this case, the model uses three fully connected layers with dropout and ReLU activation functions between them. The size of the linear layers is reduced by one-quarter throughout the network. All models finish up with a softmax layer. Figure 4.5 shows the two variants.

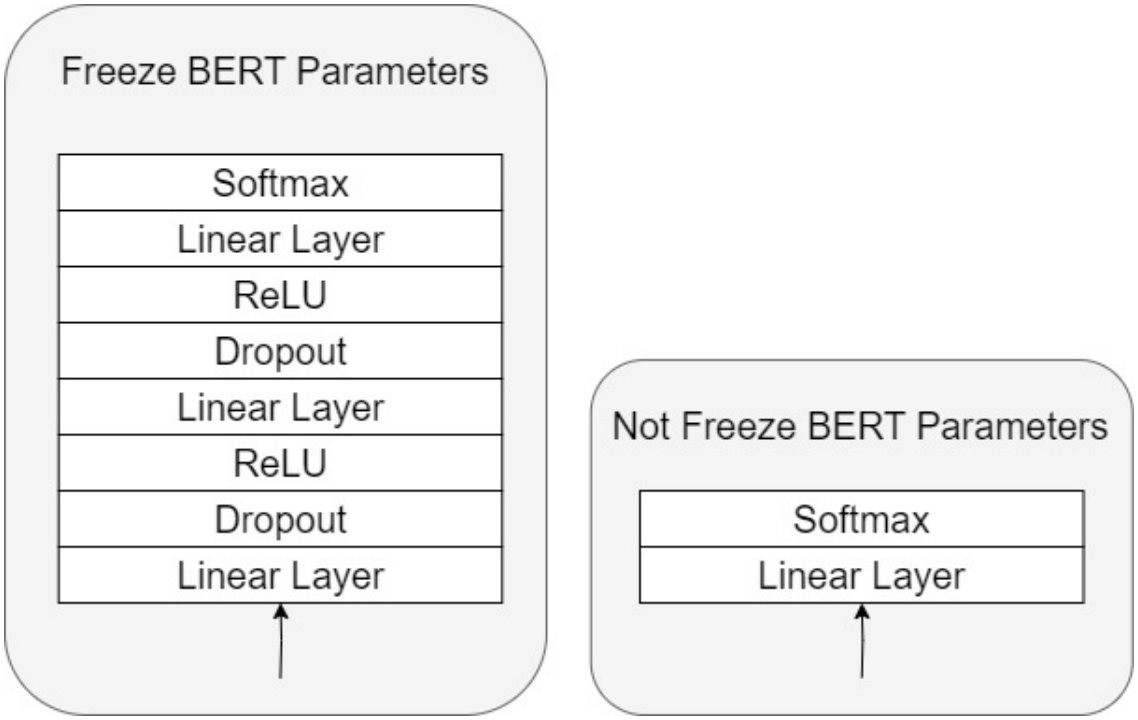

Figure 4.5: Two variants of the feed-forward component. The one to the left applies when the BERT parameters are frozen, whereas the one to the right is not. We apply a larger feed-forward component when freezing the BERT parameters because of focusing on updating the neural network parameters instead. The output from the BERT model is fed as input to the feed-forward component from the bottom to the final softmax layer.

Table 4.1 shows a list of possibly tunable hyperparameters that could be used.

Hyperparameters - BERT-based models	
Hyperparameter	**Potential Values**
Text Feature Combination	QA, QTA
Cleaning	True, False
Learning Rate	$\in \mathbb{R} : [0.001, 0.00001]$
Batch Size	$\in \mathbb{Z} : [1, 256]$
Min-Max Abstract Length	$\in \mathbb{Z} : [0, 1000]$
Dropout	$\in \mathbb{R} :< 0, 1 >$
Freeze	True, False

Table 4.1: List of hyperparameters relevant to all BERT-based models.

Chapter 5

Results

The results section presents the findings obtained from the diverse methods described in the method chapter. To evaluate their performance, a thorough exploration of different hyperparameters was conducted to maximize the model's performance on unseen data.

5.1 Training

Initially, TF-IDF models with logistic regression were employed, utilizing a learning rate of 0.0001 and a batch size 16. Two variations were evaluated, one with text cleaning and another without. The training process spanned 20 epochs. Figure 5.1 show the training performance throughout a series of epochs. The hyperparameters considered for this model were using QA or QTA as text combination as well as with and without cleaning.

The name of the models is defined in the following manner:

$$'\{text\_feature\_combination\} - \{model\_name\} - lr - \{lr\} - clean - \{cleaning\_type\} - batch - size - \{batch\_size\} max - abs - len - \{max\_abs\_len\} - dropout - \{dropout\_value\}',$$

where the fields within the curly brackets are hyperparameters.

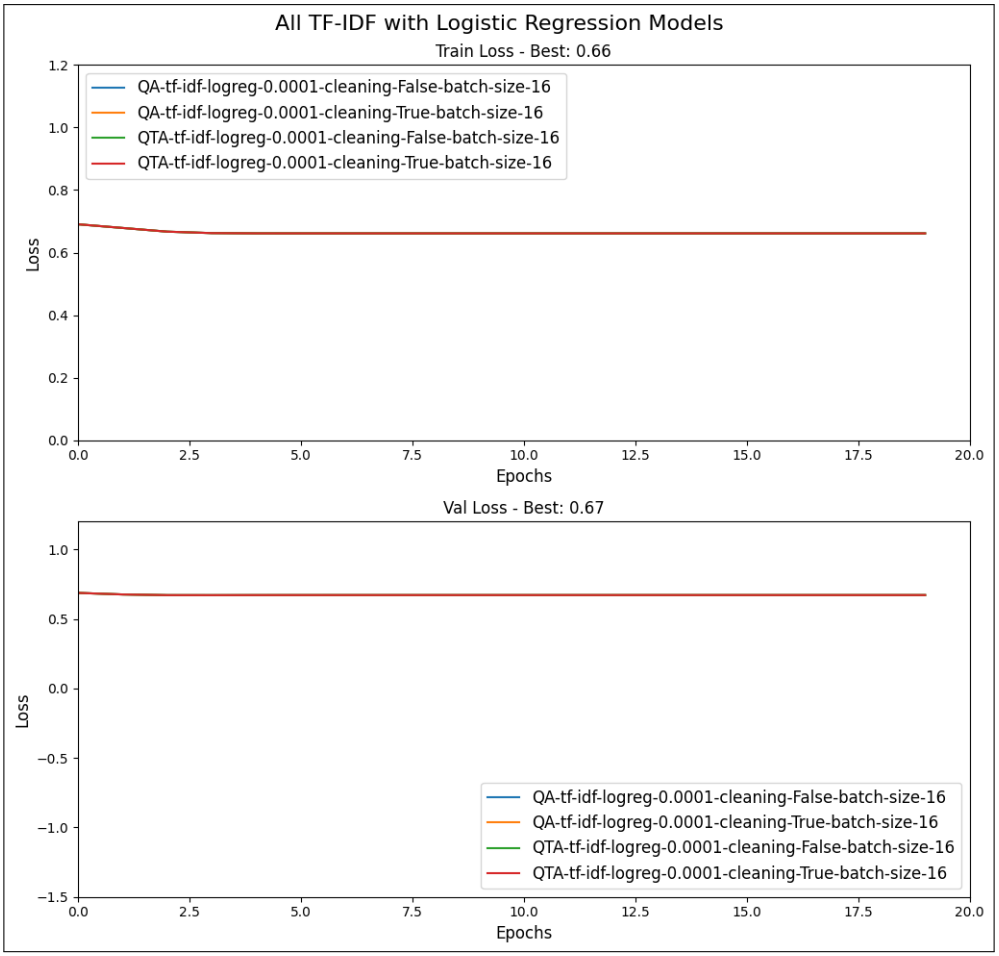

Figure 5.1: This plot showcases the performance of various variants of the TF-IDF with a logistic regression model, with each line representing a different combination of hyperparameters. The x-axis represents the number of epochs, while the y-axis represents the loss. The upper plot demonstrates the training performance, while the lower plot illustrates the validation performance. You cannot clearly see all model performances because they behave the same and thus are plotted on top of each other.

Next is the initial training sequence of the different BERT-based models. Here 8 different models were trained. Figure 5.2 show the performance of the models on both training and validation data. To distinguish the different architectures along with hyperparameters each model is given a name.

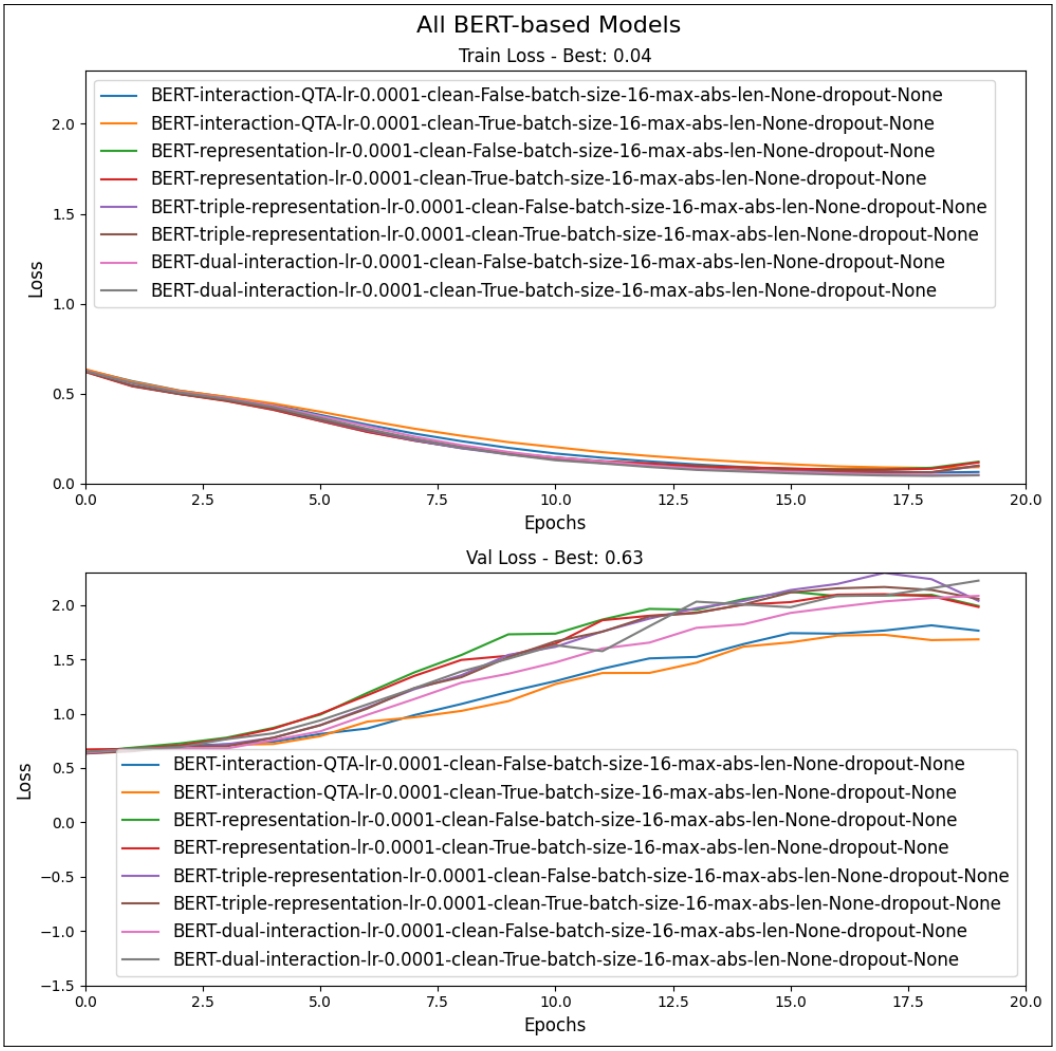

Figure 5.2: This plot illustrates the loss performance of the different BERT-based approaches, showcasing a series of models with different hyperparameters in the same plot. The x-axis represents the number of epochs, while the y-axis displays the loss values.

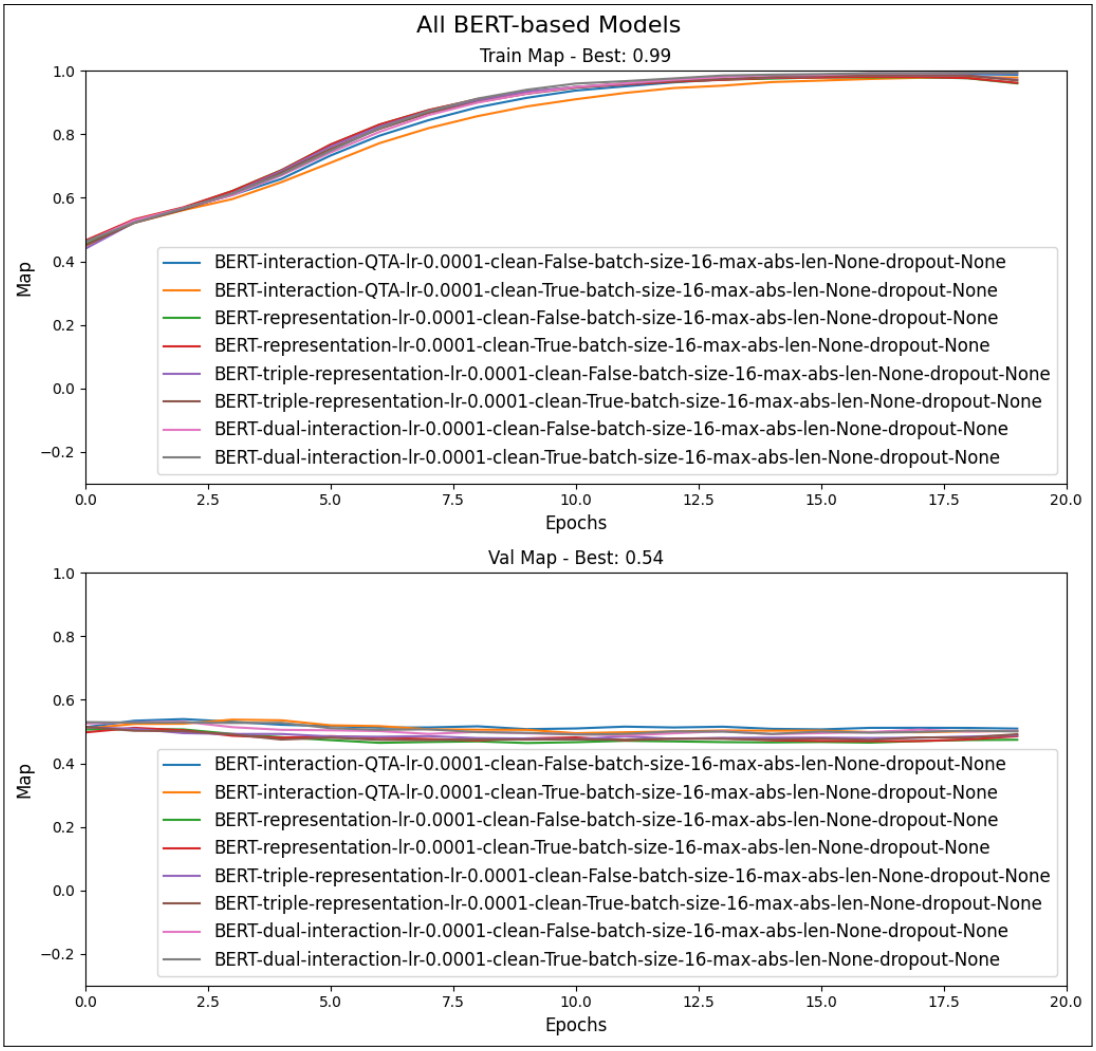

Figure 5.3: This plot illustrates the MAP performance of the different BERT-based approaches, showcasing a series of models with different hyperparameters in the same plot. The x-axis represents the number of epochs, while the y-axis displays the MAP values.

Based on the insights gained from the first iteration of training BERT-based models, knowledge was gained to fine-tune the hyperparameters in order to improve the performance of the models in iteration 2. The actions were to modify the abstract's learning rate, batch size, and maximum abstract length. The learning rate was reduced, the batch size was increased, and the abstract length was set to avoid truncating texts after they are tokenized. Figure 5.4 show the results of loss performances of different models.

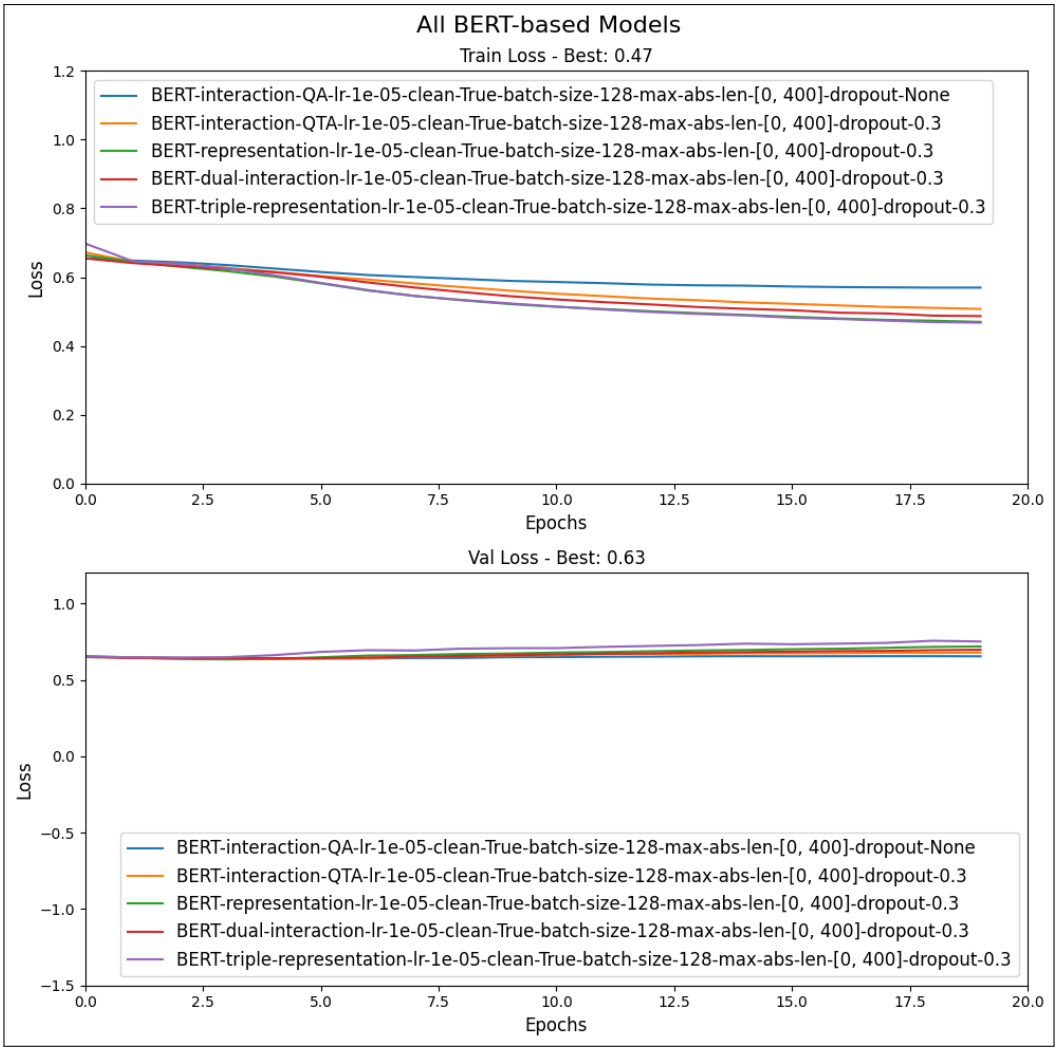

Figure 5.4: This plot visualizes the performance of different variants of the BERT models, showcasing two subplots of losses over a series of epochs. Each subplot represents the training and validation loss, respectively.

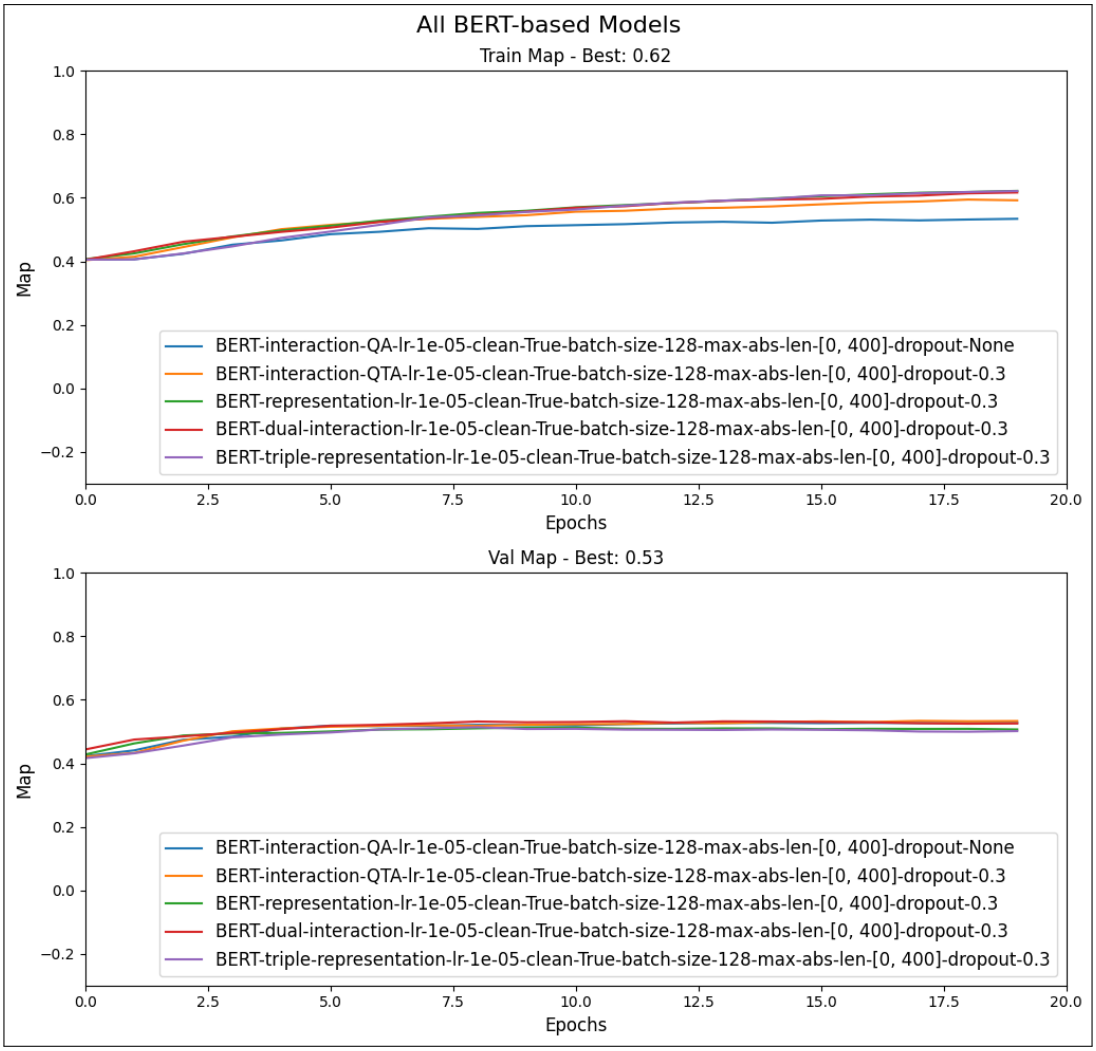

Figure 5.5: Visualization of MAP performance of different variants of the BERT models, showcasing two subplots of losses over a series of epochs. Each subplot represents the training and validation loss, respectively. The labels show the hyperparameters used.

More hyperparameter tuning was done based on the results from the training iteration 2, shown in 5.4. Figure 5.6 show the training and validation performance of different BERT-based models where the main change from earlier is that the BERT parameters are frozen.

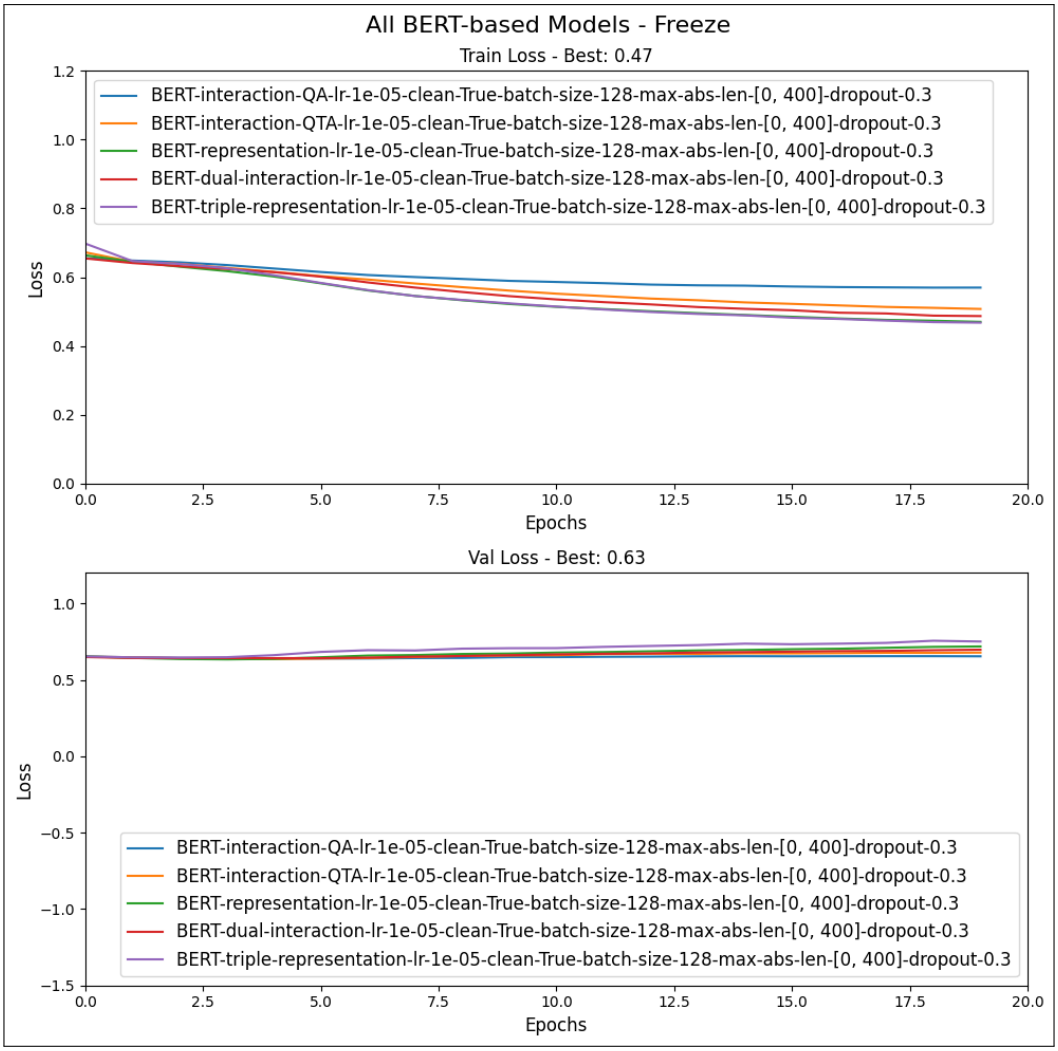

Figure 5.6: This plot visualizes the performance of different variants of the BERT models where the BERT parameters are frozen. The plot shows two subplots of losses over a series of epochs. Each subplot represents the training and validation loss, respectively.

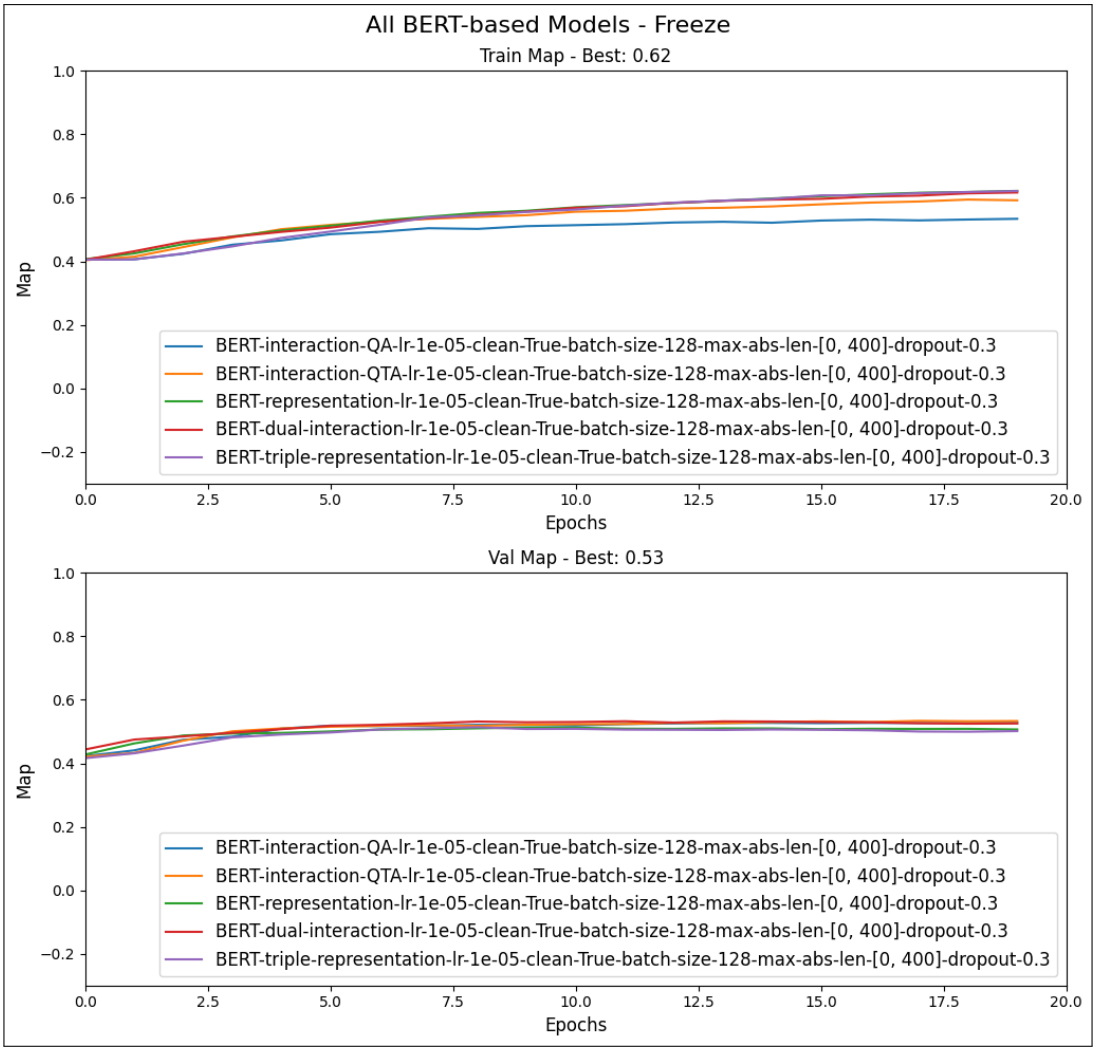

Figure 5.7: This plot visualizes the MAP performance of different variants of the BERT models where the BERT parameters are frozen. The plot shows training and validation in separate subplots over a series of epochs.

5.2 Model Selection

Model selection is based on how well the different models perform on the unseen validation data. I choose the model that scores the best validation MAP score. Table 5.1 shows the base models along with the BERT models scoring best on validation data in each training iteration. Don't exchange the definition of training iteration and epochs. Training iteration is a fully completed training session of models throughout all defined epochs.

Method	MAP	Accuracy
Base Models		
BM25	0.42	0.19
PubMed Ranker	0.04	0.006
Iteration 1		
BERT-interaction	0.54	0.66
Iteration 2		
BERT-interaction	0.53	0.66
Iteration 3		
BERT-interaction	0.50	0.67

Table 5.1: Results from the two basemodels, BM25 and PubMed Ranker. Each model has calculated MAP and accuracy on the validation set.

We can see that the model resulting in the highest MAP on validation data is the BERT interaction from the first training iteration of BERT models. This model was trained with the following hyperparameters:

- Text Combination = QA
- Learning Rate = 0.0001
- Cleaning = False
- Batch Size = 16
- Max Abstract Length = None
- Freeze = False

Now that we know which model to choose, let's try to understand why the different models perform as they do by retrieving some samples. Let's first consider PubMed ranker using some samples from the validation data. The model validation is done using a document-return rate of at most 1000 documents.

For this query with qid = 2183 there exists 84 non-related documents and 30 related queries. When querying PubMed it returns a list of 45 documents where only two of the documents from the returned list appear in the validation set. As explained in 4.2, we assume that documents returned by PubMed and not included in our dataset are having a ground truth label of 0. Thus for this example, we have 43 documents does not exists in our dataset and are treated as having a ground truth label of 0. Table 5.2 shows the the returned documents from PubMed but only including the ones existing in our dataset.

Qid	PubMed Rank	Label
2183	8	1
2183	10	0

Table 5.2: Sample of ranked documents using PubMed Ranker. The result shows the returned documents for the query with qid equal to 2183 that exists in the validation set.

Looking at another sample of evaluating a query from the validation set, a similar behavior occurs. For the query with qid = 2860, 48 documents with ground truth label = 1 and 47 with truth label = 0 exist. The PubMed search engine returns a document list of 106 documents, where 4 of the documents exist in our dataset. Table 5.3 shows the returned documents when querying qid 2860, only showing the ones included in the validation set.

Qid	PubMed Rank	Label
2860	21	0
2860	50	1
2860	71	0
2860	105	0

Table 5.3: Sample of ranked documents using PubMed Ranker. The result shows the returned documents for the query with qid equal to 2860 that exists in the validation set.

Continuing with the document with qid equal to 2860, let's see how the BERT-based model makes predictions with the best MAP score on the validation set. Figure 5.4 shows the top 10 ranked results along with the ground truth labels.

Qid	BERT-interaction-rank	Label
2860	1	1
2860	2	1
2860	3	1
2860	4	1
2860	5	0
2860	6	0
2860	7	0
2860	8	1
2860	9	1
2860	10	0

Table 5.4: Sample of ranking all documents for a particular query using Interaction-BERT. The figure shows the top 10 ranked documents for the query with qid equal to 2860.

5.3 Model Evaluation

The chosen model is now evaluated on the test set where its MAP and accuracy performance is calculated. The results are shown in Figure 5.5. The hold-out test set has a size of 6922 datapoints. The model's performance on this set represents an estimate of how well the model is expected to perform on new, unseen data in real-world scenarios.

Method	**MAP**	**Accuracy**
BERT-interaction	0.51	0.62

Table 5.5: Table of the chosen models' performance on the test set. The model's score on MAP and accuracy are calculated.

Chapter 6

Discussion

6.1 Base Models

Starting the discussion chapter, let's first analyze and discuss the findings of the base models, namely the BM25 and PubMed Ranker. Based on the MAP score obtained from PubMed, it is evident that the ranking performance is exceptionally low, reaching as low as 0.04. This signifies a poor quality of ranking.

A primary factor contributing to this poor performance is the significant number of documents in the test set that PubMed did not return. This deficiency in document retrieval severely impacts ranking accuracy. However, when considering the accuracy metric, there is a slight improvement due to the labeling approach used, where the top 35% of the results are labeled as 1 and the remaining as 0. There is also an assumption that the actual value of the documents not found in the dataset has a label equal to 0, thus resulting in a higher accuracy relative to the MAP score.

6.2 BERT-Based Models

Based on the training and validation performed on the graphs in Chapter 5 it is observed that all the models exhibit similar behavior. The trend performs exceptionally well on the training data but demonstrates poor performance on the validation data. Several factors could contribute to this behavior.

One possible reason is the difference in the tokens and patterns used in the training data compared to the validation data. There might be substantial variations between these two datasets, leading the models to learn specific patterns that are only prevalent in the training data. As a result, when faced with unseen or validation data, the models struggle to generalize their learned patterns effectively.

Another factor to consider is the similarity in text content between related and non-related documents. It is often observed that the text content in both types of documents is quite similar. This similarity poses a challenge, as the models find it difficult to discern whether the entire article contains relevant information solely based on the information provided in the abstract. This suggests that medical study authors may write the related and non-related document abstracts in the same fashion, making it challenging to accurately interpret the relevance of the whole article based on abstract information alone.

Furthermore, it is essential to highlight that BERT-based models have demonstrated impressive capabilities in sentiment analysis tasks, where identifying positive or negative sentiments in a text is relatively straightforward. However, when labeling medical abstracts as relevant or non-relevant to a systematic review title, the task becomes significantly more complex. The challenge arises from the fact that sentiment analysis relies on the explicit sentiment expressed within the text, whereas determining the relevance of a medical study to a research question goes beyond the information provided in the abstract alone.

A possible limitation of choosing the tiny-BERT is that it is only trained on general data [13]. Health data may contain different words and ways of writing than the data it is trained, resulting in a more difficult task of fine-tuning it. The tiny-BERT has a fixed vocabulary, meaning that words appearing in training data and not in the vocabulary will be regarded as unknown tokens. A possible solution for further work would be to train the BERT-based models on pre-trained BERT models such as Bio Clinical BERT [11]. This model has 12 encoders and a hidden size of 768, which is considerably larger than tiny-bert and would have taken longer to train.

A comprehensive exploration of various hyperparameters, such as learning rate, batch size, text cleaning, maximum abstract lengths, and parameter freezing, was conducted in the context of the BERT-based models. The results consistently revealed a trend of lower training loss but higher validation loss, thus overfitting. I would, therefore, not suggest further empirical experiments on hyperparameter optimization.

A crucial finding arising from these results is that using the title of the systematic review as the input query for the PubMed search engine proves to be far from optimal. Thus, relying on the systematic review title alone does not yield satisfactory document retrieval and ranking results.

6.3 Dataset

A big part of this study is related to creating a dataset that can be used to train document ranking models optimized for a particular type of user need. The study shows how it is possible to create a labeled dataset from a collection of systematic reviews by utilizing the open-source PubMed library to fetch additional information, such as the abstract of documents. Based on the results, we can see that it is possible to fine-tune existing pre-trained language models on this dataset, although with a lacking generalization performance. Therefore, it has to be questioned whether the textual information in the abstract of a particular medical document is sufficient when determining the document's relation to a systematic review title. The work on dataset creation serves as an initial point to train document ranking for the task. A richer dataset with additional features from each medical document is still to be researched.

6.4 Future Work

As mentioned, there is no need to experiment on the available hyperparameters in this study. I would rather see an implementation of the ensamble method K-fold cross-validation, a technique used to evaluate a machine learning model's performance and generalization ability. It involves dividing the dataset into k subsets, training the model on k-1 folds, and evaluating its performance on the remaining fold. This process is repeated k times, with each fold serving as the validation set, and the results are averaged to provide a robust estimation of the model's performance.

Since this study mainly focuses on training neural ranking models using the LTR framework point-wise training, other LTR framework approaches are still to be examined. These approaches come with the additional work of creating a suitable dataset. For the pairwise approach, each data point would include an input query, two medical documents (one more relevant than the other), and a binary label. The model would then predict

the relative order of the documents. This could be developed upon my dataset creation model by combining a positive and negative data point for a particular query. The listwise approach would make less sense because it requires a labeled dataset of a relative ranking of all associated documents for a particular query. And by keeping in mind that, in reality, the end user determines all evaluated documents as either related or non-related, it doesn't make sense to label a dataset of relative order.

Moving forward, an important next step would be to do more research on creating a richer dataset of more features from the medical document. This dataset could facilitate developing models that incorporate additional meta-data alongside raw text. Meta-data could include information such as author names, citations, the number of tables, or the presence of numerical features. By integrating these contextual factors, the model could gain a more comprehensive understanding of the relevance of medical abstracts to systematic review titles.

Furthermore, exploring the development of a neural network that combines the contextualized representation of the text with numerical features could prove beneficial. This hybrid approach would leverage the rich semantic information captured by BERT-based models and the informative nature of numerical features, leading to a more holistic and accurate relevance assessment. Previous research has defined an effective package for training models using transformers and tabular data [14].

A future work when the ranking algorithm works as expected is to develop information retrieval algorithms that extract relevant information from the relevant studies. Usually, the information needed in these studies is found in the article itself. Thus, the whole text of the study needs to be analyzed. Named Entity Recognition (NER) or Topic Modelling (TM) are possible extraction techniques that could be suitable. NER is a natural language processing technique that identifies and extracts specific named entities, such as people, places, organizations, and other types of entities, from unstructured text. NER is typically performed using machine learning models trained on annotated text data to recognize and classify named entities accurately. Thus, I would have to consider using a pre-trained NER model or collecting an annotated dataset. TM is a machine learning technique that identifies topics or themes in documents or text data. It works by analyzing the frequency and distribution of words within the documents and grouping them into clusters of related topics. This method does not require any labeled dataset and could be applied directly to collecting unstructured documents.

6.5 Objectives and Contributions

The method employed, and its corresponding results successfully achieve a substantial portion of the objectives outlined in this thesis. To tackle objective (A), the background chapter highlights the existing theoretical methods available for document ranking, including the related work. The study explains in-depth how the contextualized BERT language model works and how it can be used as a binary classifier. The LTR framework and the related work section describe how treating a binary classification problem as a ranking problem that suits the task at hand is possible.

Objective (B) is fulfilled by utilizing a collection of systematic reviews where the main work concerns fetching additional textual data from the PubMed library. The algorithm for creating the dataset carefully selects the most likely correct document avoiding errors in the data.

Based on the related work, the study is implementing various types of BERT-based models as well as TF-IDF with logistic regression. Additionally, the BM25 and PubMed Ranker is implemented as models for comparison. The solution facilitates training, evaluating, and selecting the best-performing model using the MAP metric. Storing the different model's performances during training and evaluation makes it possible to compare when finished. This part is thus answering objective (C).

Objective (D) is handled in the data creation part and how the training framework is developed. The data creation algorithm fetches both document titles and abstracts, and the solution makes it possible to train various models with different text feature combinations. The training framework also facilitates specifying different hyperparameters for training the models. Since each model is stored by adding the different hyperparameters in the model name, it is easy to compare their training and validation performances in a single plot.

The discussion, solution, and results chapters tackle objective (E), which concerns analyzing the results and performances of the models. The discussion highlights the strengths and weaknesses of why the result turned out as it did.

When considering the work done in this study along with the related work presented in 2.5, clear contributions can be found. BM25 model still serves as a base model that can reduce the number of documents by filtering out the most non-related documents. This work applies many of the same methodologies as passage re-ranking with BERT.

Similarities include using BERT as an interaction-based model for reranking and adding a task-specific layer. The main difference concerns training the model on another dataset. This work also contributes to the related work that uses representation-based ranking with BERT, concatenating the contextualized representation and adding a task-specific layer instead of using a comparison score. In addition, I am highlighting the limitations of existing tools for systematic reviews, especially focusing on PubMed Ranker. The query samples drawn in the model section show that only a small portion of the true labeled documents (in our dataset) is fetched from PubMed, even by setting a high limit for the documents returned. Figures from the results show an example of this limitation where out of 106 documents returned, only a single document is regarded as related and occurs in the 50th position of the ranking. Imagine the time of screening 49 non-related documents before reaching a related document.

Finally, let us contextualize this work within the problem highlighted in the introduction. The issue revolves around minimizing the time spent screening irrelevant documents when conducting a systematic review of a specific medical intervention. While the algorithms proposed in this research do not necessarily rank all relevant documents at the top, this may not always be essential. Particularly in time-sensitive situations, such as during the initial phase of a pandemic, policymakers face the challenge of making rapid decisions, leaving little time for comprehensive document screening. Nevertheless, employing an algorithm capable of ranking documents at high speed with a reasonably accurate assessment can be advantageous. This approach facilitates fast evidential analysis of relevant documents, prioritizing those most likely to yield valuable insights. Although my solution may not fully meet the performance criteria for ranking, it serves as a stepping stone for leveraging trained ranking models.

Chapter 7

Conclusion

This thesis aims to determine how existing document ranking algorithms can be trained to assist researchers conducting systematic reviews. The findings show that training many document ranking algorithms is possible by treating the problem as a binary classification task. The research shows how to create a labeled dataset from a collection of systematic reviews by retrieving additional textual features from PubMed. Further, the research develops a document ranking framework including base models such as BM25 and PubMed ranker, TF-IDF with logistic regression, and a series of BERT-based models. The BERT-based models take advantage of existing pre-trained models and fine-tune them by adding task-specific linear layers.

A big part of the research then focuses on utilizing the training framework to train several types of BERT-based models with different hyperparameters. The results show that all BERT-based models can learn patterns in the training data but struggle to generalize to perform well on unseen data. These findings show significant differences in the training and validation data resulting from the model's ability to learn the patterns of 80268 data points but failing to perform well on a hold-out set of 7633. The results show that the abstract of medical content does not provide sufficient information to regard its relevance to a systematic review title.

A possible reason for overfitting is the dataset used to train the document ranking algorithms. More research must create a richer dataset that includes more content from the actual document, not only the abstract. Leveraging all textual, numerical, tables, and listings from the medical document opens up many opportunities for combining BERT-based models and other non-language models.

Given the available resources, the research process could not have been done differently regarding the type of LTR framework used and how the dataset was created. However, I am critical to my choice of developing a series of BERT-based models when it turned out that they all behaved quite similarly. After developing the Interaction-BERT model, it would be better to research other pre-trained models and task-specific layers and experiment with other hyperparameters related to the training procedure. This could, for example, include other types of optimizers, K-fold cross-validation, and adjustable learning rate.

To sum up, this research provides a first-ever attempt at training existing pre-trained language models on a custom dataset created from a collection of systematic reviews and PubMed. It also provides deep research on training these models using the pointwise binary prediction training approach. The developed training framework facilitates training the implemented models on a new type of data as well as using the existing data format to train new models.

Bibliography

[1] Amin Nakhostin-Ansari Seyed Hossein Hosseini Asl Mehrnush Saghab Torbati Reyhaneh Aghajani Zahra Maleki Ghorbani Shahriar Faghani Amir Valizadeh, Mana Moassefi. Abstract screening using the automated tool rayyan: results of effectiveness in three diagnostic test accuracy systematic reviews. 2022.
URL: https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-022-01631-8.

[2] SAIMA SADIQ-SALEEM ULLAH SEYEDALI MIRJALILI ANAM YOUSAF, MUHAMMAD UMER. Emotion recognition by textual tweets classification using voting classifier (lr-sgd). 2020.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9309291.

[3] Niki Parmar-Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. 2017.
URL: https://arxiv.org/pdf/1706.03762.pdf.

[4] Harikrishnan N B. Confusion matrix, accuracy, precision, recall, f1 scorek. 2019.
URL: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd. Accessed: 2023-05-25.

[5] Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in nli: Ways (not) to go beyond simple heuristics. 2021.
URL: https://arxiv.org/abs/2110.01518.

[6] Saidoung P Blaizot A, Veettil SK. Using artificial intelligence methods for systematic review in health sciences: A systematic review. 2022.
URL: https://pubmed.ncbi.nlm.nih.gov/35174972/.

[7] Andriy Burkov. The hundred-page machine learning book. 2019.

[8] Cochrane Community. Covidence.
URL: https://community.cochrane.org/help/tools-and-software/covidence. Accessed: 2023-05-25.

[9] Shane Connelly. Practical bm25 - part 3: Considerations for picking b and k1 in elasticsearch. 2018.
URL: https://www.elastic.co/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch.

[10] Carlos Lopezosa 1 Cristòfol Rovira, Lluís Codina. Language bias in the google scholar ranking algorithm. 2021.
URL: https://www.mdpi.com/1999-5903/13/2/31/html.

[11] Willie Boag Wei-Hung Weng Di Jin Tristan Naumann Matthew B. A. McDermott Emily Alsentzer, John R. Murphy. Publicly available clinical bert embeddings. 2019.
URL: https://huggingface.co/emilyalsentzer/Bio_ClinicalBERT.

[12] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Understanding the behaviors of bert in ranking. 2019.
URL: https://arxiv.org/pdf/1810.04805.pdf.

[13] Viktor Karlsson. Tinybert — size does matter, but how you train it can be more important. 2021.
URL: https://medium.com/dair-ai/tinybert-size-does-matter-but-how-you-train-it-can-be-more-important-a5834831fa7d.

[14] Akshay Budhkar Ken Gu. A package for learning on tabular and text data with transformers. 2021.
URL: https://aclanthology.org/2021.maiworkshop-1.10/.

[15] Cochrane Library. About the cochrane library. .
URL: https://www.cochranelibrary.com/about/about-cochrane-library. Accessed: 2023-05-25.

[16] Cochrane Library. Cochrane database of systematic reviews. .
URL: https://www.cochranelibrary.com/cdsr/about-cdsr. Accessed: 2023-05-25.

[17] Cochrane Library. About cochrane reviews. .
URL: https://www.cochranelibrary.com/about/about-cochrane-reviews. Accessed: 2023-05-25.

[18] Ihab Ilyas Hossam Hammady Madian Khabsa, Ahmed Elmagarmid and Mourad Ouzzani. Learning to identify relevant studies for systematic reviews using random forest and external information. 2015.
URL: https://link.springer.com/article/10.1007/s10994-015-5535-7.

[19] eal R. Haddaway Michael Gusenbauer. Which academic search systems are suitable for systematicreviews or meta-analyses? evaluating retrieval qualities ofgoogle scholar, pubmed, and 26 other resources. 2018.
URL: https://onlinelibrary.wiley.com/doi/epdf/10.1002/jrsm.1378.

[20] Brian D. Davison Jeff Heflin Mohamed Trabelsi, Zhiyu Chen. Neural ranking models for document retrieval. 2021.
URL: https://link.springer.com/article/10.1007/s10791-021-09398-0.

[21] Zbys Fedorowicz Ahmed Elmagarmid Mourad Ouzzani, Hossam Hammady. Rayyan—a web and mobile app for systematic reviews. 2016.
URL: https://systematicreviewsjournal.biomedcentral.com/articles/10.1186/s13643-016-0384-4.

[22] University of Bergen. Fairchoices dcp analytics tool. University of Bergen, 2022.
URL: https://www.uib.no/en/bceps/130756/fairchoices-dcp-analytics-tool. Accessed: 2023-05-25.

[23] Ministry of Health-Ethiopia. Health sector transformation plan ii. pages 9–11. Global Financing Facility, 2021.
URL: https://www.globalfinancingfacility.org/ethiopia-health-sector-transformation-plan-201920-202425. Accessed: 2022-12-10.

[24] World Health Organization. The global health observatory. .
URL: https://www.who.int/data/gho/data/themes/topics/indicator-groups/indicator-group-details/GHO/sdg-target-3.8-achieve-universal-health-coverage-(uhc)-including-financial-risk-protection. Accessed: 2023-05-25.

[25] World Health Organization. What we do. .
URL: https://www.who.int/about/what-we-do. Accessed: 2023-05-25.

[26] World Health Organization. Universal health coverage. .
URL: https://www.who.int/health-topics/universal-health-coverage#tab=tab_1. Accessed: 2023-05-25.

[27] PubMed. Pubmed overview.
URL: https://pubmed.ncbi.nlm.nih.gov/about/. Accessed: 2023-03-15.

[28] Sudharsan Ravichandiran. Getting started with google bert. Packt Publishing, 2021.

[29] Kyunghyun Cho Rodrigo Nogueira. Passage re-ranking with bert. 2020.
URL: https://arxiv.org/pdf/1901.04085.pdf.

[30] Google Scholar. Google scholar about. .
URL: https://scholar.google.com/intl/en/scholar/about.html. Accessed: 2023-05-25.

[31] Google Scholar. Google scholar inclusion. .
URL: https://scholar.google.com/intl/en/scholar/inclusion.html#crawl. Accessed: 2023-05-25.

[32] Aman Shrivastav. Gaussian error linear unit (gelu). OpenGenus.
URL: https://iq.opengenus.org/gaussian-error-linear-unit/. Accessed: 2023-05-25.

[33] Ross DA Smith PG, Morrow RH. Field trials of health interventions: A toolbox. 3rd edition. 2015.
URL: https://www.ncbi.nlm.nih.gov/books/NBK305514/.

[34] Susan Sykes. How to solve your top systematic review challenges. National Library of Medicine, 2019.
URL: https://blog.distillersr.com/how-to-solve-your-top-systematic-review-challenges. Accessed: 2023-05-25.

[35] Shan Jin Debin Liu Ye Wang, Zhi Zhou and Mi Lu. Comparisons and selections of features and classifiers for short text classification. 2017.
URL: https://iopscience.iop.org/article/10.1088/1757-899X/261/1/012018/pdf.

[36] Chenyan Xiong Yifan Qiao. Understanding the behaviors of bert in ranking. 2019.
URL: https://arxiv.org/pdf/1904.07531.pdf.