FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Applying Learning to Rank methods to small datasets
Student:	Bc. Adriána Majtánová
Supervisor:	Ing. Martin Labský, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Learning to Rank methods (L2R) are used to improve relevance ranking within various kinds of result lists.
Most commonly, L2R is used to rank web search results given a user query as well as information known about the user.
L2R combines traditional Information Retrieval methods such as TF/IDF search with Machine Learning methods aimed at improving
the ordering of the initial result lists delivered by the baseline method.

L2R use cases range from textual search to recommender systems.
For search tasks, training data is typically acquired in the form of query/relevant document list pairs.
L2R has proven to work well on large datasets.
It has also been applied to smaller, domain-dependent datasets, e.g. sorting Slack chat search results by message relevance.
How much L2R improves over baseline is however much less known and documented for these smaller datasets.

Goals of the thesis:
- Research state-of-the-art L2R algorithms, available metrics and datasets.
- Choose an appropriate dataset for experiments, such as:
  - Web Answer Passages https://ciir.cs.umass.edu/downloads/WebAP/
  - MS MARCO https://microsoft.github.io/msmarco/

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 21 June 2022 in Prague.

- Implement an L2R-enabled Information Retrieval system that combines a baseline IR system with supervised learning
  - the baseline IR system will most likely use Apache SOLR, Apache Lucene, or Indri
  - the baseline IR system may additionally use Sentence Transformer models https://www.sbert.net/ and perform a semantic search by looking up similar embedding vectors
  - pick an L2R algorithm and try to improve the baseline IR system's results
- Quantify whether and how L2R contributes to result relevance over the baseline IR system using training datasets of progressive size
  - the training datasets of progressive size can be created by subsampling a single original dataset
  - show how L2R affects the relevance of retrieved results for varying sizes of training data, e.g. 500, 1k, 10k, 20k training pairs

References:
Tie-Yan Liu: Learning to Rank for Information Retrieval. Foundations and Trends in IR, Vol. 3, No. 3 (2009) 225–331
Luis Sanchez et al: Easing Legal News Monitoring with Learning to Rank and BERT. https://link.springer.com/chapter/10.1007/978-3-030-45442-5_42
Shuguang Han et al: Learning-to-Rank with BERT in TF-Ranking. https://arxiv.org/pdf/2004.08476.pdf

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Applying Learning to Rank methods to small datasets

Bc. Adriána Majtánová

Department of Applied Mathematics
Supervisor: Ing. Martin Labský, Ph.D

May 4, 2023

Acknowledgements

First and foremost, I would like to thank my thesis supervisor Ing. Martin Labský, Ph.D for his advice, support and willingness to answer all my questions throughout the process of writing this thesis. His feedback was very helpful and I am grateful I can keep improving my skills under his guidance. I would also like to thank my family, friends and especially my partner Vojtěch for their endless encouragement.

Declaration

Citation of this thesis

Majtánová, Adriána. *Applying Learning to Rank methods to small datasets.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Dostupnosť veľkého množstva trénovacích dát sa stalo neoddeliteľnou súčasťou vytvorenia kvalitného modelu využívajúceho strojové učenie. Pri použití strojového učenia v reálnych aplikáciach však mnohokrát nastáva problém s datasetmi, nakoľko ich veľkosť nie je dostatočná. Táto práca sa zaoberá uplatnením tzv. Learning to Rank (LTR) metód na malé datasety. V rámci práce skúmame efektivitu LTR modelu LambdaMART v porovnaní s tradičnými systémami na vyhľadávanie informácií. Vyhodnotili sme množstvo rôznych experimentov na 2 rôznych datasetoch – dataset MS MARCO a český dataset DaReCzech. Experimenty sú navrhnuté tak, aby odpovedali na tri výskumné otázky. Otázky sa zameriavajú na porovnanie základného systému, ktorý využíva vyhľadávanie na základe termov oproti systému využívajúcemu Learning to Rank, ďalej potrebným počtom trénovacích dát a aký vplyv na výsledky má nahradenie základného systému za predtrénovaný transformer model. Dosiahnuté výsledky ukazujú, že LTR metódy dosahujú očividné zlepšenie oproti obom verziám základného systému, a to aj napriek použitiu obmedzeného množstva dát.

Klíčová slova Learning to Rank, vyhľadávanie informácií, MS MARCO, DaReCzech, LambdaMART, transformer

Abstract

The availability of large datasets has become almost indispensable necessity for building a well-performing machine learning model. However, many real-world applications, especially those in specialised domains, offer only data limited in size. This thesis is devoted to appplication of Learning to Rank (LTR) methods to small datasets. In this thesis we explore the effectiveness of LambdaMART, a powerful LTR model, compared to a traditional Information Retrieval (IR) system. We conduct a series of multiple experiments on two different datasets, the MS MARCO dataset and a dataset in Czech language called DaReCzech. The experiments are designed with the purpose to answer three research questions. The questions focus on comparing a baseline term-based IR system with an LTR-enabled system system, the amount of training data needed, and the impact of replacing the term-based first-stage retrieval with pre-trained sentence transformer models. Our findings show, that LTR methods achieve a notable improvement over both types of baseline, even with the constrained training data.

Keywords Learning to Rank, information retrieval, MS MARCO, DaReCzech, LambdaMART, transformer

Contents

List of Figures

List of Tables

Introduction

With the rapid increase in the amount of data shared online, search systems have become crucial in helping users quickly access the information they are looking for. An essential part of modern information retrieval systems is the ability to prioritise search results based on the user's input. Traditional IR systems typically rely on a fixed function that scores documents against a user's query. Such scoring functions are usually based on the term frequency and inverse document frequency components and favor documents with highest word overlaps with the query. Scoring each document then yields a final ranked list of search results for the query.

Although traditional IR systems work well in many scenarios, they have several drawbacks. Since they use a predefined set of rules or algorithms to produce the score, they do not have the ability to learn from user feedback and continuously improve their performance. Additionally, many systems are only keyword-based. Although they are efficient and the computation of scores is relatively fast and easy, they can be limiting when it comes to understanding the meaning and the context of sentences. A different solution for addressing the ranking problem is to use machine learning.

Learning to Rank (LTR) is a supervised machine learning task, where the objective is to construct a model capable of predicting the importance or relevance of a set of objects with respect to the given query and sort them accordingly. The most common application of Learning to Rank algorithms is found in the field of information retrieval. LTR algorithms can help to improve the relevance of the results returned to users by traditional IR systems. In contrast to traditional IR systems, Learning to Rank addresses this task by using machine learning models trained on labelled ranked data. Even though LTR has been shown to be effective on large datasets, in real-world scenarios the training data is often limited. The effectiveness of LTR algorithms on smaller domain-specific datasets can be challenging and still needs to be explored further.

Goals of the thesis

This thesis aims to find the answers for the following three research questions, marked as RQ1,RQ2 and RQ3:

RQ1 This research question is dedicated to comparing a baseline Information Retrieval system with a LTR-enabled IR system. We aim to quantify *whether and how sorting the documents retrieved from the baseline system using pre-trained ranking model contributes to the result relevance.*

RQ2 The second research question is an extension of the previous question by using varying sizes of training data. Assuming that the LTR model outperforms the baseline IR system, our goal is to determine *how much (or how little) training data is needed for the model to improve the baseline results.* We experiment with data of different sizes obtained by subsampling the original training dataset used in RQ1.

RQ3 Additionaly, we aim to experiment with improving the baseline by adding pre-trained sentence transformer models. The objective is to analyse *how the overall performance of our multi-stage ranking pipeline changes opposed to the one using keyword approaches only.*

Thesis organisation

Chapter 1 provides an introduction to information retrieval and its traditional ranking methods. It also introduces performance measures used to evaluate models in IR. Chapter 2 reviews the fundamental background of Learning to Rank methods, their classification and existing algorithms. It also explains the multi-stage ranking system. Chapter 3 analyses datasets suitable for the LTR task and the examples of features that are commonly used in this domain. Chapter 4 describes the experimental setup and implementation – the technologies used, the data processing, the process of feature extraction and model training are described in this chapter. Finally, Chapter 5 presents the results of the proposed experiments, their analysis and the answers for the research questions defined above.

CHAPTER 1

Ranking in Information Retrieval

The term information retrieval refers to the process of obtaining information from a collection of data. Requests for information (usually provided by the user of a search system) are called queries, and a collection of data or documents is called corpus. Depending on the application, the data in the corpus can consist of textual documents, images, audio files, or other items. This thesis focuses on documents that are entities that contain text. This chapter also analyses the most frequently employed performance measures for IR. The approaches to IR differ, starting from simple Boolean model to more complex methods such as vector space and probabilistic models. The subsequent sections delve into the main ideas behind the most widely adopted conventional ranking models.

1.1 Boolean model

The earliest and simplest IR systems were built on Boolean algebra and set theory. Documents were typically represented as binary vectors of terms, where each term was assigned a binary value that indicated its occurrence in the given document. To specify a query, users needed to construct a combination of logical operators (AND, OR, NOT) between terms. Boolean model stood out by its simplicity and could be efficiently implemented using an inverted index, however, there were many limitations to this solution – the model was not able to predict relevance scores, it could only determine whether the document was relevant to the given query, therefore, ranking the search results based on relevance was not possible (optionally, it was possible to sort the resulting documents according to the document age or other document feature) [1].

1.2 Vector space model

Vector space model depicts both documents and queries as vectors in high-dimensional space. The values in the vector belong to the weights of terms and, owing to the fact that the queries and vectors have the same representation, it is possible to measure the distance between the two. This distance can then be interpreted as a dissimilarity or the relevance of the document to the query. The vector model is highly parameterizable, meaning different distance measures and approaches to compute term weights can be used. One of the most widely used weighting schemes is called TF-IDF (term frequency-inverse document frequency) [2]. TF stands for term frequency of a term t_i in document d_j (typically normalised), and IDF is defined as

$$\mathrm{IDF}\,(t_i) = \log\frac{|D|}{d(t_i)}, \tag{1.1}$$

where $|D|$ refers to the total number of documents in the collection, a $d(t_i)$ is the number of documents that contain the term t_i. By multiplying term frequency with its inverse document frequency for each term, one can create the document vector. The common weight to measure the similarity between the query and document vector is to compute the cosine similarity (or distance). The cosine similarity of two vectors u and v is defined as follows

$$\mathrm{cos\_sim}(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||}. \tag{1.2}$$

After querying, the result is a ranked list of documents ordered by their similarity to the query. The vector space model is a simple geometric model, which can be efficiently implemented; its advantages over standard Boolean model are that it provides a continuous relevance value between queries and documents and therefore allows ranking according to this value. Nevertheless, VSM assumes many independent low-level terms, it is not able to deal with documents having similar context which use different term vocabulary and the term weights in the vector do not take the original order of terms into consideration.

To avoid the assumption of term independence, some extensions of the vector model were suggested, such as Latent Semantic Indexing (LSI) [3]. This model uses singular value decomposition (SVD) to preprocess the feature space. After the transformation, the document is represented by concepts (which are linear combinations of terms) instead of terms. By constructing a new semantic space, where terms with similar context are associated, LSI can overcome the issues with synonyms that VSM has. It also reduces the dimensionality of the feature space. On the other hand, SVD algorithms have a high complexity and the preprocessing of the feature matrix can be an expensive operation.

1.3 Probabilistic model

Another group of ranking models for Information Retrieval are models based on the probabilistic ranking principle [4]. The probabilistic ranking principle claims that in an IR system, where for each query there is a list of documents sorted by decreasing probability of their relevance to the query, if these probabilities are estimated as accurately as possible, the system will reach the best obtainable effectiveness. The goal is to determine the probability of relevance of the document d to the query q, $P(r|d,q)$. Ranking documents by given probability is equivalent to ranking according to log-odds of that probability:

$$\log(O(r|d,q)) = \log\left(\frac{P(r|d,q)}{1-P(r|d,q)}\right) = \log\left(\frac{P(r|d,q)}{P(\overline{r}|d,q)}\right). \tag{1.3}$$

By applying Bayesian and chain rules and other simple transformations to the expression shown above it is possible to derive $\log\left(\frac{P(d|q,r)}{P(d|q,\overline{r})}\right)$, which is the core of all probabilistic models. The individual probabilistic models then differ in the way how $P(d|q,r)$ is computed [1].

1.3.1 BM25

One of the most famous probabilistic models is BM25. Given a query q with terms $t_1,\ldots,t_m$, BM25 is defined as:

$$\mathrm{BM25}(d,q) = \sum_{i=1}^{m} = \frac{\mathrm{IDF}(t_i)\cdot\mathrm{TF}(t_i,d)\cdot(k_1+1)}{\mathrm{TF}(t_i,d)+k_1\cdot\left(1-b+b\cdot\frac{\mathrm{len}(d)}{\mathrm{avgdl}}\right)}, \tag{1.4}$$

where k_1 and b are free parameters of the model and *avgdl* is the average length of document in the collection. The parameter k_1 calibrates the document frequency scaling (when set to 0 we get a binary model, and on the contrary, large values correspond to using the raw term frequency). The variable $b \in [0,1]$ controls the scaling by document length, where $b=0$ means that the length is not normalised.

1.4 Statistical language model

An example of a statistical language model is a Language model for IR (LMIR). The aim of this approach is to estimate a language model for each document d in the corpus. To create the ranking of the documents based on a query q, the language model uses query likelihood. Unlike the approach in the previous section, which focused on modelling the probability of relevance of a document d to a query q, in language models, the probability of generating the terms in the query by the document model is used. Given the query q containing terms $t_1,\ldots t_n$ and document d, the unigram LMIR assumes the independence of each term in the query. The probability of generating query q from document d is given as

$$P(q|d) = \prod_{i=1}^{n} P(t_i|d), \tag{1.5}$$

where $P(t_i|d)$ is the probability of generating term t_i from the document d. The simplest method to estimate the $P(t_i|d)$ is to count the number of times the term appears in the document divided by the total number of terms:

$$P(t|d) = \frac{\mathrm{TF}(t,d)}{|d|}. \tag{1.6}$$

1.4.1 Smoothing for Language Models

An important concept in LMIR is smoothing, which refers to the adjustment of the maximum likelihood estimator of a language model to improve its accuracy [5]. Smoothing is essential to avoid the zero probabilities that the model can produce if a term in the query does not appear in the document. Jelinek and Mercer proposed smoothing that interpolates the document model with the collection model:

$$P(t_i|d) = \lambda P(t_i|d) + (1-\lambda)P(t_i|D), \tag{1.7}$$

where D is the entire collection of documents and $\lambda \in (0,1]$ is a smoothing parameter. Setting λ close to zero is recommended for shorter queries [6]. Another popular smoothing method is Dirichlet smoothing, where the probability of generating term t_i works by adding a pseudo-count to each word in the document:

$$P(t_i|d) = \frac{\mathrm{TF}(t_i,d) + \mu P(t_i|D)}{|d| + \mu}. \tag{1.8}$$

The probability of the term in the collection of documents D is meant as a pseudo-count, and it is controlled by the smoothing parameter μ. The value of this parameter should be set depending on the document length [6], for example the default value used in Apache Lucene[1] is 2000.

1.5 Metrics

This section describes the most commonly used performance measures for evaluating the ranking models.

[1]https://lucene.apache.org

1.5.1 Mean Reciprocal Rank

For a single query q, reciprocal rank is defined as $\frac{1}{r(q)}$, where $r(q)$ is the position of the first relevant document in the ranked list. If the document with the highest relevancy does not appear in the search result list, the reciprocal rank equals zero. The mean reciprocal rank for the whole set of queries Q can be computed as

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{r(q_i)}. \tag{1.9}$$

Since MRR only takes into account the position of the first relevant document, it is a useful metric for datasets that only have one relevant result, such as question answering datasets.

1.5.2 Normalized Discounted Cumulative Gain

Discounted Cumulative Gain penalises those documents that appear lower in the result list but are highly relevant, by reducing the relevance value logarithmically proportionally to the position of the document. Let rel_i refer to the relevance score of the document at position i and let k be the number of documents considered. Then the formula of NDCG at position k is defined as

$$NDCG@k = \frac{DCG@K}{IDCG@K}, \tag{1.10}$$

where DCG is computed as

$$DCG@k = \sum_{i=1}^{k}\frac{2^{rel_i}-1}{\log_2(i+1)}. \tag{1.11}$$

$IDCG@k$ is the ideal DCG, meaning the maximum possible $DCG@k$ one can obtain given this ranking list. The normalised version of DCG is preferred due to the fact that in the ranking task, the number of documents retrieved for each query may differ, thus normalising against the ideal gain is necessary to compare the score between different queries [7].

1.5.3 Mean Average Precision

Precision at k for query q can be defined as:

$$P@k(q) = \frac{\sum_{i=1}^{k} r_i}{k}, \tag{1.12}$$

where r_i is a binary indicator showing whether document i is relevant for query q (setting $r_i = 1$). Then we are able to compute the average precision of query q at k items:

$$AP(q)@k = \frac{1}{\sum_{i=1}^{k} r_i}\sum_{i=1}^{k} P@i(q)\times r_i. \tag{1.13}$$

Then the Mean Average Precision for the entire query set Q is computed as:

$$MAP = \frac{\sum_{i=1}^{|Q|} AP(q_i)}{|Q|}. \tag{1.14}$$

This metric is designed for binary ratings as it does not consider the ranking of retrieved items. Only the presence (or absence) of relevant documents is important.

Chapter 2

Learning to Rank

Let Q be a set of n queries $Q = \{q_1, q_2, \dots, q_n\}$ and D a document set $D = \{d_1, d_2, \dots, d_m\}$. For each query q_i there exists a list of relevant documents $R(q_i) \subseteq D$ of size l_i associated with the query and a set of labels $y_i = \{y_{i,1}, y_{i,2}, \dots, y_{i,l_i}\}$. The training set is denoted as $S_{\text{train}} = \{(q_i, R(q_i)), y_i\}_{i=1}^{n}$. Typically, query-document pairs are represented by a feature vector $x_i = \Psi(q_i, d_j)$, where $i = 1, \dots, n; j = 1, \dots, l_i$ and Ψ is a feature extractor. The process of creating features and their examples is described in the next chapter in Section 4.2. The goal of the LTR task is to learn a ranking fuction $f(q, d) = f(x)$, that can assign a score to a given query document pair (their feature vector), such that when the documents are ordered by their score it reflects their relevance to the query. The notation used above was adopted from [8] with slight modifications. In the following sections, when query-document pairs are mentioned as an input, we mean their feature vector representation unless stated otherwise. This chapter explains the concept of learning to rank and reviews a part of the most commonly used algorithms. These algorithms can be classified into three categories: pointwise, pairwise, and listwise.

2.1 Pointwise methods

Pointwise approach is the oldest method used to solve LTR problems. These algorithms treat each document from the training set individually – the method predicts the relevance of a single document for a given query, instead of ranking the whole list of items. The document relevance can be represented by a real-valued score, a categorical score (which can be either binary meaning relevant vs. irrelevant document, or into multiple categories), or ordinal score. The goal is to predict the score for each document-query pair.

The advantage of this approach is that any traditional machine algorithm can be used in the pointwise solution, since the ranking task can be reformulated as a classification or regression task. The input space is query-document pairs and the target label is the relevance of the document y.

One drawback of this approach is that the model does not utilise the position of each of the documents in the list of matching documents for the query, and loosing this valuable information may result in suboptimal performance. Furthermore, explicit labels are needed for each training data point, and as described in the Chapter 3 the process of collecting relevance judgments can be complicated. After training, the model can predict the relevance of new query-document pairs it has not encountered before, and in the end it is possible to create a resulting ranked list of relevant documents based on the predicted score.

2.1.1 Algorithms

Some example of common algorithms that adopt the pointwise approach include the following methods. PRank [9] is a neural ranking algorithm that uses a linear function called perceptron to estimate the relevance score of document d_i. The output of the model is $f(d_i) = w^T x_i$, where w is a trained vector of vector of weights. The learning in PRank is driven by stochastic gradient descent.

An additional example of a pointwise algorithm is MART [10] which stands for Multiple Additive Regression Trees. Regression tree is a decision tree algorithm that is used to predict continuous valued outputs [11]. In MART, the final model is a linear combination of the outputs of singular regression trees. It can be interpreted as boosting algorithm that is performing gradient descent in function space, using regression trees [12]. The final model can be written as:

$$F_N(x) = \sum_{i=1}^{N} \alpha_i f_i(x), \tag{2.1}$$

where $f_i(x) \in R$ is a function modelled by a single regression tree, $\alpha_i \in R$ denotes the weight associated with the i_{th} tree and x is a feature vector.

2.2 Pairwise methods

Pairwise methods tackle the ranking problem as a pairwise classification task. The input space consists of document pairs and the objective is to determine whether the first document is more relevant to the given query than the second. The model learns how to score a pair of documents so that the more relevant document is ranked higher. The advantage of the pairwise approach is that the model is using the rank information in the training set, even though only in a pairwise manner. Also, no explicit labels for each document-query pair are needed, only pairwise preference is necessary.

2.2.1 Algorithms

Some example pairwise algorithms include RankBoost [13], RankSVM and RankNet [14]. RankBoost is a method based on AdaBoost [15], the only difference is that it uses the distribution defined on document pairs instead of individual documents as in AdaBoost [16]. RankBoost learns a ranking function by combining multiple weak rankers into a strong ranker.

RankSVM uses SVM for pairwise classification, the objective function is the same as in SVM and the mathematical formulation only differs in the constraints, which are constructed from the document pairs. RankSVM minimises the hinge loss function. The following subsection delves into RankNet in more detail, as it played a significant role in LTR development.

2.2.2 RankNet

According to [16], RankNet was presumably the first LTR algorithm used by commercial search engines. It uses a neural network as an underlying model, but any model that produces a differentiable function of the model parameters can be used [12]. For a given query and a pair of documents represented by their feature vectors x_i and x_j with a different label, the model computes the scores $s_i = f(x_i)$ and $s_j = f(x_j)$. Let $d_i \gg d_j$ define the document d_i being more relevant than the document d_j. The output of the model is mapped to a learned probability that d_i should be ranked higher than d_j through a sigmoid function with parameter σ determining its shape:

$$P_{ij} \equiv P(d_i \gg d_j) \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}}. \tag{2.2}$$

RankNet uses the cross-entropy cost function to penalise the deviation from the true probability $\overline{P}_{ij}$ of document d_i being ranked higher than document d_j:

$$C = -\overline{P}_{ij}\log P_{ij} - (1 - \overline{P}_{ij})\log(1 - P_{ij}). \tag{2.3}$$

Assumed that the true probability is known as $\overline{P}_{ij} = \frac{1}{2}(1 + S_{ij})$, where $S_{ij} \in \{-1, 0, 1\}$ is defined to be 1 if document d_i is more relevant than document d_j, 0 if they are equally relevant and -1 otherwise, then the cost can be specified as:

$$C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log(1 + e^{-\sigma(s_i - s_j)}). \tag{2.4}$$

RankNet learns via gradient descent – the model weights are updated using the gradient of the cost with respect to the model weights. The gradient of the cost with respect to the model scores s_i is used in order to specify the gradient with respect to the model weights.

2.3 Listwise methods

These methods aim to learn a ranking function that can order a set of items in a meaningful way for a given query, based on the overall relevance of the items for the query. Listwise methods benefit from utilising all information about document position in the list. Generally, there are two categories of listwise methods based on how they optimise a ranking function on the whole document list. The first option is to directly optimise the IR evaluation measure (as the measures discussed in Section 1.5). The other approach consists of minimising a surrogate loss function, such as the probability loss function defined on permutations [17].

2.3.1 Algorithms

The first listwise example is LambdaRank [18], a model based on RankNet described above (2.2.2). The algorithm builds upon the idea that for training the model, the costs themselves are not needed, only their gradients with respect to the model scores. LambdaRank aims to write down the desired gradients directly rather than deriving them from a cost [12]. The intuition is that specifying the rules that determine how the rank order of documents should change after sorting them by score is easier than constructing a general smooth optimisation cost [18]. Lambdas (λs) in the model can be understood as indicators of the directions in which documents should move in the ranked list in order to optimise performance.

ListNet [19] and ListMLE [20] are common instances of listwise learning to rank algorithms from the second category. These approaches use the loss function that measures the difference between the output model and ground truth permutations. ListNet is founded on the idea of minimising the KL-divergence between the predicted ranking and the true ranking of the items. Using the Luce model, ListNet defines the permutation probability distribution based on the scores given by its scoring function, which is compared with permutation probability distribution based on the ground truth label. It employs neural network model and train using the gradient descent. ListMLE is an improvement of ListNet that uses the negative log likelihood of ground truth permutation as the ranking loss, which reduces the training complexity compared to ListNet [16].

2.3.2 LambdaMART

The model [12] was developed as a combination of MART and LambdaRank algorithms described in the previous sections. Instead of a neural network, LambdaMART uses gradient boosting to optimise the cost function. Gradient boosting is a technique that uses an ensemble of weak learners.

```
parameters: $N$ trees, $m$ training samples, $L$ leaves per tree, learning rate $\eta$
1  for $i = 0$ to $m$ do
2      $F_0(x_i) = \text{BaseModel}(x_i)$ // set to 0 for empty BaseModel
3  end
4  for $k = 1$ to $N$ do
5      for $i = 0$ to $m$ do
6          $y_i = \lambda_i$
7          $w_i = \frac{\delta y_i}{\delta F_{k-1}(x_i)}$
8      end
9      $\{R_{lk}\}_{l=1}^{L}$ // $L$ leaf tree on $\{x_i, y_i\}_{i=1}^{m}$
10     $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ // Assign leaf values based on Newton step
11     $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate $\eta$
12 end
```

Algorithm 1: Pseudocode summarising the LambdaMART algorithm [12].

Since IR performances measures such as NDCG (1.5.2 or MAP (1.5.3) are not differentiable at all points and thus cannot be used directly as loss functions, LambdaMART uses the idea with λs from LambdaRank. Simply put, LambdaMART just needs MART, to specify gradients (which are the lambdas in this case) and the Newton step [21] to work. For every given pair of documents d_i and d_j, where $d_i \gg d_j$ and the documents have already been sorted by score, the λ_{ij}, which can be interpreted as force that will push the document in the list upwards/downwards, is defined as follows:

$$\lambda_{ij} = \frac{-\sigma|\Delta Z_{ij}|}{1 + e^{\sigma(s_i - s_j)}}. \tag{2.5}$$

Z_{ij} represents the utility difference generated by changing the rank positions of documents d_i and d_j, common example is to use NDCG as Z. While LambdaRank updates all the weights after each query, LambdaMART only updates a few parameters at a time, but using all the data, because the splits at the nodes are computed using all the data that falls to that node. A pseudocode for better understanding of the algorithm taken from the original publication can be viewed in 1. The pseudo code and the process of building LambdaMART model can be summarised to the following steps:

1. Create a base model F_0.

2. For each iteration k (of N trees), construct a tree for all query-document pairs x:

a) Update the documents scores using F_{n-1}.

b) Compute the λ and w – the derivative of λ.

c) Fit a next tree with L leaves. For each leaf node l assign the value γ_l, based on the λ of the instances that reach the node l. Do a Newtons step to optimize gradient prediction in terminal nodes.

d) Update the model F_k.

3. The result is the model F_N

In 2010, an ensemble of LambdaMART models won Track 1 of the 2010 Yahoo! Learning To Rank Challenge and became the state-of-the-art Learning to Rank model at the time. Even more than 10 years after that, LambdaMART still beats neural approaches on multiple benchmark Learning to Rank datasets with numerical, hand-crafted features [22, 23].

2.4 Recent deep learning approaches

In the recent years, several neural ranking models have been introduced. A survey on deep learning ranking models from 2019 [24] shows that some of them outperform the traditional LTR algorithms that use hand-crafted features. One of the first significant models to do so is DeepRank [25] proposed in 2017, which simulates the human judgment process. The model simulates the process in three steps, first by detecting the relevant locations by extracting query-centric contexts. After that, the local relevances between the queries and each query-centric context are determined using convolutional neural network or two-dimensional gated recurrent units. The last step is to aggregate local relevances to a global one using recurrent neural network and term gating network to produce global relevance for ranking.

Another example is a HIerarchical Neural maTching model (HiNT) [26] introduced in 2018, which consists of two stacked components. The first is local matching layer used for modelling the semantic matching between a query and each passage of a document. This is followed by the global decision layer that accumulates local signals to determine the final relevance score.

Neural re-ranking solutions benefit from no need for an extra feature engineering phase as part of data preparation, as they are able to work with raw texts directly, where determining the important parts of the inputs are part of the model's architecute. On the other hand, the traditional LTR models offer better interpretability and depending on the task, using own set of features may be required.

2.4.1 Transformers

With the introduction of BERT (Bidirectional Encoder Representations from Transformers) [27] in 2018, transformers have become state-of-the-art in many NLP tasks, including text classification [28], machine translation [29], language modeling [30], sentiment analysis [31], question answering [29] and many others. Unlike recurrent neural networks, which are another alternative to processing sequential data, transformers have the benefit of processing the sequences in parallel.

The key advantage of transformers in IR, or NLP in general, is the ability to capture the semantic meaning of words and sentences. Transformers achieve this by using the powerful attention mechanism, based on pairwise matching of individual parts of the input. Attention allows the model to discover the relationship or importance of each word to the other parts of the input sentence. Introduced by [32], the original transformer's architecture consists of an input layer, an encoder, a decoder and an output layer. The encoder and decoder are both composed of multiple stacked encoder (or decoder) blocks. Each block accepts and outputs a sequence. The blocks consist of multi-head self-attention layer and fully connected neural network layers.

Application in Information Retrieval

In the context of information retrieval, there are two high-level categories of strategies to incorporate transformers [33]. The first one is the most straightforward application, where the transformers are trained to directly output the relevance score. This is often done using dual-encoder architecture, called Siamese network [34]. The task is to determine a probability of how the given document is relevant to the given query and then sort the documents accordingly based on this probability score. This approach is typically part of a multi-stage ranking architecture, where a smaller pool of documents is retrieved first by a keyword engine (e.g. using BM25) [33]. The transformer then serves as a re-ranker.

The other option is to leverage the learned dense representations. The vector representation obtained from the transformer model captures the semantic relations well and can replace sparse features, which rely mostly on term matching. The ranking task can then be looked at as the problem of approximate nearest-neighbour search, where the objective is to find the most similar vectors to the vector that represents the query. A frequently employed measure for evaluating the similarity of embeddings is the cosine similarity described in 1.2 [33].

Limitations

On the contrary, transformer models are limited by their large size. Training a transformer model can be extremely resource intensive. Even though the model is capable of processing sequences in parallel, the model still requires significant memory and computational resources. Compared to competing architectures, transformers are very data-hungry – large amounts of training data are needed for their optimal performance. Although transformers can be adapted to individual domains by applying transfer learning, a sufficient amount of data is still needed for the fine-tuning part. With small datasets, the model may overfit and perform suboptimally. Also, BERT-based models are not able to handle long input documents, and longer documents either need to be truncated or split into smaller chunks to obtain the proper representations.

2.5 Learning to Rank in practice

Learning to Rank methods have been proven to work well on a set of various benchmark datasets [35], however, in real-life applications it is not possible to determine the relevance score for all documents in the corpus to construct the final list of top k most relevant documents. To balance search efficiency and effectiveness in IR systems, LTR models are usually part of a multi-stage architecture [36]. Figure 2.1 shows the components of such architecture.

2.5.1 Multi-stage ranking pipeline

Since the whole collection of documents is typically very large, the first-stage retrieval system is needed to return an initial subset of candidate documents (for example, of size 100–10000). First-stage retrieval should optimally achieve high recall efficiently, the goal being to return a pool of documents that contain as many relevant documents as possible, without taking too much time. After that, the re-ranking stage is employed – this stage can consist of one or multiple re-rankers that gradually decrease the size of the possibly relevant documents. Since in this stage only a smaller set of documents is examined, re-rankers are usually more sophisticated models. Finally, the sorted top k documents (for example, 10) are retrieved to the user.

2.5.2 Term-based first-stage retrieval

In terms of first-stage retrieval, term-based models have been dominating this area for a long period of time [36]. Especially the BM25 ranking model has demonstrated effective performance in various research works [37, 38]. Term-based models have the advantage of efficiency, they are typically employed using inverted index, and they do not require heavy computational operations to manage large-scale collections of documents. On the other hand, since they match strictly based on terms, without taking the term ordering into account, they may struggle with capturing semantic context of documents and suffer from vocabulary mismatch problem. If high recall error is introduced in the early stage of ranking pipeline, the re-ranking models may not be able to

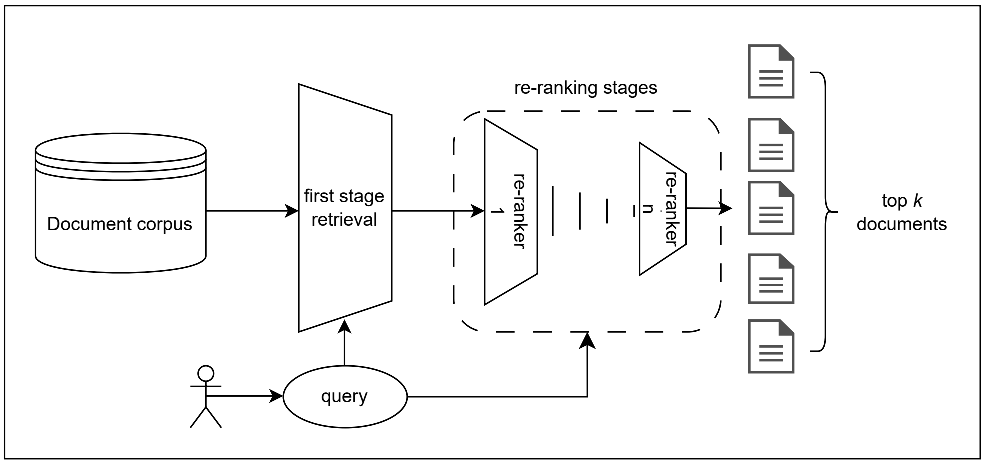


Figure 2.1: The multi-stage architecture of information retrieval system.

improve the score, as they were not presented enough relevant documents in the beginning.

2.5.3 Semantic first-stage retrieval

An alternative to term-based solution for the first-stage retrieval is to use semantic retrieval models. There have been several efforts by researchers to use dense retrieval methods for retrieval [36]. The requirements for the dense model are that the document representation should be independent of the query, so it can be pre-computed. For a real-time application, the computation of the query representation and the final layer that determines the final score need to be as simple as possible.

A viable method for semantic retrieval is to apply semantic search over the learnt dense representations for documents using nearest neighbour search. A naive approach would be to use k-Nearest-Neighbour search (kNN), for example with cosine similarity as defined distance function. However, kNN requires comparing the query vector to every vector in the document collection. The complexity of kNN grows linearly with the number of items, making exact search impossible to use with large datasets. To reduce the computation complexity, Approximate Nearest Neighbor search (ANN) was introduced. ANN algorithms sacrifice part of precision in favor of efficiency gain. The existing ANN algorithms can be classified into multiple categories, such as tree-based, graph-based, or hash-based.

Chapter 3

Data

The acquisition of suitable data for LTR is a challenging task, partially due to the need for a list of relevant judgments. To use certain ranking models, it is necessary to collect the relevance for each document-query pair. Relevance judgements can be either accumulated manually by an expert panel or automatically derived from logging user behaviour.

The manual evaluation by experts may be error prone or biased depending on the evaluator and their different interpretation or perception of the query and the documents. As evaluating each document in the collection is infeasible, multiple search systems are typically used to create a pool of documents, which are then manually inspected, ranked, and assigned a corresponding label.

For automatically constructed labels, one example would be to use user clicks – the approach is built on the idea that the more relevant documents should be clicked more often. However, this also brings a certain bias to the data, as users have the tendency to click on the results that appear higher in the search results list, even though the first result does not necessarily have to be the most relevant one [39]. On the other hand such implicit feedback is less costly to collect and can be collected in large quantities compared to the manual feedback. Furthermore, recent research attempts to adapt LTR models to be able to overcome the issues with bias, a successful example is [40].

This chapter reviews some of the publicly available information retrieval datasets and datasets commonly used in the learning to rank domain. We focus on datasets containing the original fulltexts for both queries and documents. The characteristics, advantages and limitations of each dataset are further analysed and discussed.

3.1 MS MARCO

MS MARCO [41] stands for MAchine Reading COmprehension dataset. It is a set of datasets used in the domain of natural language processing released in

Set type	Number of queries	Number of queries filtered
Train set	808,731	502,939
Dev set	6,980	6,980
Eval set	6,937	–

Table 3.1: The number of unique queries available for training, validating and evaluating MS MARCO dataset.

2016. The dataset comprises of multiple collections suitable for different tasks such as document retrieval, passage retrieval, document re-ranking, question answering, key-phrase extracion and others. The queries were created by anonymising and sampling Bing's search logs and the documents were retrieved from Bing using its large-scale web index. From these web documents, the passages were automatically extracted and then labelled by human evaluators. Document and passage retrieval are the tasks that receive the most attention from the researcher community. The tasks on the official website[2] are defined as follows:

1. Document Re-Ranking: Given a candidate top 100 document as retrieved by BM25, re-rank documents by relevance.
2. Document Full Ranking: Given a corpus of 3.2M documents generate a candidate top 100 documents sorted by relevance.
3. Passage re-ranking: Given a candidate top 1000 passages as retrieved by BM25, re-rank passage by relevance.
4. Passage Full Ranking: Given a corpus of 8.8M passages, generate a candidate top 1000 passages sorted by relevance.

Additionally both document and passage retrieval have an active leaderboard[3] where researchers can publish their solution and compare MRR scores.

3.1.1 Dataset properties

The size of the passage ranking dataset is more than 8.8M passages derived from 3.2M documents and 300K queries. The passages (and the documents) do not have different degrees of relevance judgements; the passage can be either relevant (1) or irrelevant (0) for the given query. On average, there is one relevant document per query, and some queries from the dataset have no relevant passages at all. The passage ranking task also offers data in the form of triplets consisting of query id, relevant passage ID and irrelevant

[2]https://microsoft.github.io/msmarco/

[3]https://microsoft.github.io/MSMARCO-Document-Ranking-Submissions/leaderboard/

Field	Value
Query ID	1006683
Query	which cells contain chloroplasts
Relevant passage	Chloroplasts are organelles present in plant cells and some eukaryotic organisms. Chloroplasts are the most important plastids found in plant cells.It is the structure in a green plant cell in which photosynthesis occurs. Chloroplast is one of the three types of plastids.hloroplasts are organelles present in plant cells and some eukaryotic organisms. Chloroplasts are the most important plastids found in plant cells.
Irrelevant passage	The presence of chloroplasts in the guard cells mean photosynthesis will occur. This leads to an accumulation of glucose in daylight and the guard cells will absorb water by o . . . smosis from the epidermal cells which lack chloroplasts and do not produce glucoes.n fact, they are the only epidermal cells that have chloroplasts that can do this. Usually, we think of photosynthesis as by Mesophyll cells. There is some theory that the manufacturing of sugar can work in conjunction with K+ to regulate water potential.

Table 3.2: An example of query and its pair of passages from the MS MARCO triplets dataset.

passage ID. Negatively labelled passages were selected from a list of the top 1000 passages scored using BM25. Table 3.2 presents an example from this dataset. The dataset is already split into training, development and evaluation set. Table 3.1 shows the number of queries in the given sets; the filtered version denotes the number of queries that have at least one relevant passage. There are different subsets available; here we state the small versions of the dev and eval set, which are created from 6.8 % of queries available in the full version. Since the leaderboards are still active, the ground truth for evaluation data is not available.

3.1.2 State-of-the-art

In January 2019, the passage ranking task saw a notable increase in its state-of-the-art with the first application of transformers to a ranking task presented in [42]. This was a pivotal moment for the information retrieval community, as it led to the new era of research, now dominated by the transformer architecture [43]. Since then, BERT-based approaches have consistently held the top spots on the leaderboard. As of April 2023, both the current best

Set	#queries	#query-document pairs	avg #docs per query
Train big	175,992	1,431,730	8.1
Train small	1,852	97,386	52.6
Dev	793	41,220	52.0
Test	2,323	64,466	27.8

Table 3.3: DaReCzech statistics.

solutions on the passage re-ranking [44] and the full ranking task [45] utilise the BERT model in their solution.

3.2 DaReCzech

DaReCzech [46] is the Czech text relevance ranking dataset created by Seznam.cz. The dataset is already divided into 4 disjunctive subsets: train-big, train-small, dev, and test set. Table 3.3 shows the number of records in each of these dataset parts and the average number of labelled documents for each query. All data comes in the format of query-document pairs, where each document is composed of *title*, *URL* and *doc* field. The *doc* field contains following three subfields: *title*, *URL* and *bte*. Each query-document is annotated with a relevance label representing one of four degree levels (*Useful, A little useful, Almost not useful* and *Not useful*).

The authors originally intended the train-big set to be used for training neural network and the train-small for gradient boosted trees. They train their deep learning model built on top of Czech Electra model on the train-big set and select the best version on the dev set. Then they use the output of this model as an additional feature to gradient boosted regression trees ranker (GBRT) owned by Seznam, which is trained on the new feature with other 575 privately owned features that have been fine-tuned and improved in the production over the years. The GBRT model is trained on the train-small dataset.

As the data is not publicly available, we do not attach an example of the queries or the documents. The dataset can be requested from Seznam.cz and it is dedicated for academical, research, non-commercial purposes only.

3.3 NFCorpus

NFCorpus [47] is a full-text Learning to Rank dataset for the medical information retrieval. The 3,244 queries of various lengths are written in simple English, while the documents are composed of titles and abstracts of scientific articles in the medical domain with a highly technical vocabulary. The dataset also contains 169,756 relevance judgments, which were created from direct and indirect links from the website NutritionFacts.org. The relevance judgements were automatically extracted in the following way: the most relevant documents correspond to the medical publications that were cited in the article on the website (the query), the second-level judgements are documents that are internally linked through another NutritionFacts article that has a direct link to the publication and the lowest level is used for the pairs that are only connected through topics or tags on the website. The dataset is free to use for academic purposes.

3.4 Other datasets

There are other popular and benchmark datasets worth mentioning used in the Learning to Rank field such as Yahoo! Learning to Rank Challenge Datasets [48] and datasets provided by Microsoft – LETOR 3.0 [49], LETOR 4.0 and MSLR-WEB [50]. The reason why these datasets were not further considered in the experiments is that they do not contain the original full-text documents used to create the document query feature vectors. Since we aimed to use our own representation of documents in this thesis, enabling us to bring the semantic factor into the feature vectors, the datasets with already precomputed features could not be used. However, unlike in the Yahoo dataset, the features in datasets provided by Microsoft are not fully anonymised, and the description of most of the features is provided, meaning the data can be used as a useful source of inspiration for feature creation. The authors of [51] provide an insightful analysis of 136 features of the MSLR-WEB dataset and study their importance for multiple LTR models.

3.5 Features

The extraction and selection of features is a crucial step in the ranking pipeline. We divide the features into 3 categories:

3.5.1 Query-dependent document features

Query-dependent features are extracted from both the query and the document.

Term-based features These features include the frequency (TF) of feature terms in the document, the cover ratio of the query, the different relevance scores obtained for each term in the query, and others. Such features are usually computed per whole document, and additionally for the documents containing different fields also field-wise (e.g. a set of features is added for a document title, document body, and so). The aggregate of term-based features is also frequently used by applying statistics such as min, max, mean, median, variance, and sum of these signals.

Score features This category of features includes score features from existing retrieval models. The most commonly used features are scores from BM25, language models with different types of smoothing, TF-IDF scores, and others.

Transformer-based features There are multiple ways in which signals from transformer neural networks such as BERT can be employed. The first approach is to directly use the embeddings produced by the model as features, as the authors in [45] did. The concatenated query-document pairs with a separator character between the two were fed into the BERT model, and then the pooled output was used as an input for the ranking model. Another option is to use the pre-trained transformer reranker to output a probability that a given document is relevant to the given query. This probability score can then be used as a feature, as in [52].

3.5.2 Query-independent document features

Click-based features Examining user behaviour might be useful in determining relevance judgments. On the other hand, the number of clicks per document and the dwell time per document may serve as an important signal, given the fact that the more clicked documents should be more relevant.

Document properties The IDF score for each term in the document can be used to express the rarity or significance of the term within the document collection. This category also covers the features that describe the length of a document text, whether the entire length of the document, the length of its title, or other sections.

URL features Many datasets contain the original URL of the documents that can also contribute to the feature set. For example, the length of an URL or a number of slashes can be used. Especially in web search tasks, popular web pages tend to have shorter URLs, which are easier to remember [51].

Other web features For web search, other commonly used features are inlink and outlink numbers (the number of web pages that quote or cite particular web page and vice versa), PageRank [53] score, web page quality score, and others.

3.5.3 Query features

Query features, or document-independent features, refer to the features created from some aspect of the query without considering its relevant documents – meaning that the features have the same value across all documents associated with this query. Some examples of query features are the number of unique terms in the query, pre-retrieval performance predictor scores, likelihood of n-gram query in title fields, number of tokens, number of retrieved entities in the documents retrieved for the given queries, and more. Researchers in [54] investigated the usefulness of such queries and found that query features for the LambdaMART algorithm can improve performance.

Chapter 4

Experimental Setup

The following chapter describes the technologies and tools used for experiments, details on data selection, preparation, preprocessing, the process of feature extraction and model training.

4.1 Technologies

All experiments were implemented using Python 3.10. Vector representations from sentence transformers were computed using a system with NVIDIA GeForce GTX 1080 GPU. All remaining parts of the experiments, including feature extraction, model training, and evaluation, were performed using a computer with 4 core Intel Core i7 processo and 16GB of available RAM.

4.1.1 Elasticsearch

Elasticsearch[4] is an open-source search engine built on top of Apache Lucene[5]. It comes with a set of REST APIs, making ElasticSearch a powerful tool for handling large amounts of data. It uses an inverted index to store data and provides a wide range of search features, with the full-text search being one of the most popular. It also has a Ranking evaluation API for evaluating the quality of search results over a set of defined test queries. Moreover, it offers a Python Elasticsearch Client, which is easy to work with.

4.1.2 ir_datasets

The Python interface ir_datasets [55] is a tool used to efficiently manage popular IR datasets. It provides the option to download data, iterate through queries and documents, and search documents by their IDs effectively. The

[4]https://www.elastic.co
[5]https://lucene.apache.org

Train set name	#queries	#query-doc pairs
mini	3,113	37,356
small	7,725	92,700
medium	38,749	464,988
large	77,373	928,476

Table 4.1: The number of records in subsets created from the MS MARCO training data.

implementation works with memory inexpensive data structures such as iterators, which makes working with large amounts of data manageable. Thanks to this library, users do not have to tackle the problem of parsing different data formats, as this tool loads the data and serves them to the user using the same object representation for each dataset. It also has a concise documentation with description, citation, and metadata for each of the datasets. From the data mentioned in the previous chapter, ir_datasets holds MS MARCO and NFCorpus.

4.1.3 LightGBM

LightGBM [56] is a high-performance framework created by Microsoft. It has been shown by multiple works [57, 22] that LightGBM's LambdaMART implementation outperforms its competing libraries such as XGBoost[6] and RankLib[7] by both speed and accuracy. While most decision tree learning models grow trees level-wise, LightGBM builds trees leaf-wise (by choosing the leaf with maximum loss to grow).

4.1.4 Hugging Face

Transformers by Hugging Face [58] is a natural language processing library that provides thousands of pre-trained models. The library stores models for different tasks such as text classification, information extraction, question answering, and others. The models from Transformers can be used for over 100 languages. We use the Transformers API to download and use a pre-trained sentence model.

4.1.5 Other libraries and modules

We use pandas [59] for all data analysis and manipulation. NumPy [60] is used for mathematical computations and we also apply the matplotlib [61] science style from SciencePlots [62] for data visualisations.

[6]https://xgboost.readthedocs.io/en/stable/
[7]https://github.com/codelibs/ranklib

4.2 Data preparation

From the datasets mentioned in Chapter 3, we choose the MS MARCO and DaReCzech datasets for the experiments. The reason for this is that both datasets are of sufficient size – the number of documents they contain is in the order of millions – which is suitable for our task, since we can subsample the data for the experiments without any limitations. Furthermore, DaReCzech is in Czech language, so it offers an extra insight into how the chosen methods apply when working with different languages. Additionally, opposite to NF-Corpus, they contain mostly general, non-domain specific terms. Apart from possible issues with heavy medical terminology, NFCorpus does not contain negative labels – the non-relevant documents for the queries are not provided. Hypothetically, they could be created from all the other documents in the dataset, that were not marked as relevant for given query, but this could bring errors and biases into the training data. Unlike NFCorpus, both MS MARCO and DaReCzech contain negatively labelled query-document pairs. As for the other datasets, as mentioned previously, they do not contain the original full-text representations.

4.2.1 MS MARCO

We work with the MS MARCO passage set in the experiments.

Data engineering

To create training data, we use a small version of the original training data of the respective dataset as provided by the ir_datasets module. The small version of training data (10 % of the original size) in triplet format consists of query ID, its relevant document ID, and its non-relevant document ID for each data point. From the original training data, we choose a subset of queries. We omit queries which do not have at least one relevant result. For the majority of queries, there is only one relevant passage and up to 1,000 irrelevant passages. Since we do not need all the negatively labelled examples for the training, we randomly choose 11 irrelevant passages for each query, resulting in the final training set where for each query there exist 12 training data points (1 relevant and 11 irrelevant query-passage pairs). By subsampling the original set of queries, we create train sets of various sizes, named *mini, small, medium* and *large*. Details of the size of the data are summarised in Table 4.1.

As the relevant passages for the queries in the official evaluation set are not available, we use the small version of the development set as the test data. This set is also provided by ir_datasets. Our final test set consists of 6980 queries. For hyperparameter tuning, we use a split of the training data.

Index	Name	Feature description
1	Query length	The number of terms in the query.
2	Doc length	The number of terms in the document.
3	Covered query number	The number of query terms covered by the document.
4	Covered query ratio	Covered query number divided by the total number of terms in the query.
5-9	TF	Sum, min, max, mean and median of term frequencies for each query term.
10-14	Normalized TF	Sum, min, max, mean and median of normalized term frequencies for each query term.
15-19	IDF	Sum, min, max, mean and median of inverse document frequency for each document term.
20-24	TF-IDF	Sum, min, max, mean and median of TF-IDF score for each query term.
25	BM25	BM25 score for the whole query.
26	LM.DIR	Score from Language Model with Dirichlet smoothing, for the whole query.
27	LM.JM	Score from Language Model with Jelinek Mercer smoothing, for the whole query.

Table 4.2: The full description of the features used for the MS MARCO dataset.

Feature extraction

In the feature extraction phase, the features were created based on the features used in datasets provided by Microsoft [49] and the analysis of their usefulness [51]. Some of these features are privately owned by Microsoft and the passage ranking task does not contain different fields per document, so the features are computed per the whole document, which is composed only of the text field. We mostly focus on term-based and score features. The Table 4.2 depicts the full list of implemented features. For obtaining the term-based features, the original query texts are tokenised and the acquired terms are then preprocessed by lowercasing, removing stopwords and all non-numerical or non-alphabetical characters, and applying stemming, all performed using tools from nltk [63] library. To retrieve similarity scores for score-based features (features 25–27), three different Elasticsearch indices are used, each configured with a different similarity metric. For Language Model with Dirichlet smoothing, the smoothing parameter is set to the default value $\mu = 2000$. With Jelinek-Mercer smoothing, we leave $\lambda = 0.1$ and for BM25 we

Dataset	#queries	#query-doc pairs
small (train)	1,852	72,033
big (train)	24,178	752,542
dev	2,316	55,695
test	1,886	53,247

Table 4.3: The number of records in each set of DaReCzech after filtering.

use $b = 0.75$ and $k_1 = 1.2$. Feature extraction takes approximately 30 minutes on CPU.

4.2.2 DaReCzech

Data engineering

Since the data only comes in a format of labelled query document pairs, to employ multi-stage ranking pipeline it first is necessary to create the full set of documents. We merge the documents from all 4 sets (*train-big, train-small, dev* and *test*), assign each unique document a unique ID and create a document set of size 1,289,384. Note that we consider documents with the same title and body duplicates, even if their URL is different. We parse the *doc* field to extract the *bte* field which is further referred to as *body.* Documents with both empty title and empty body are removed from the dataset, resulting in a full dataset of documents of size 1,239,862. All documents are indexed in Elasticsearch using the same settings as for MS MARCO, except for the Elasticsearch Analyzer that is set for Czech language.

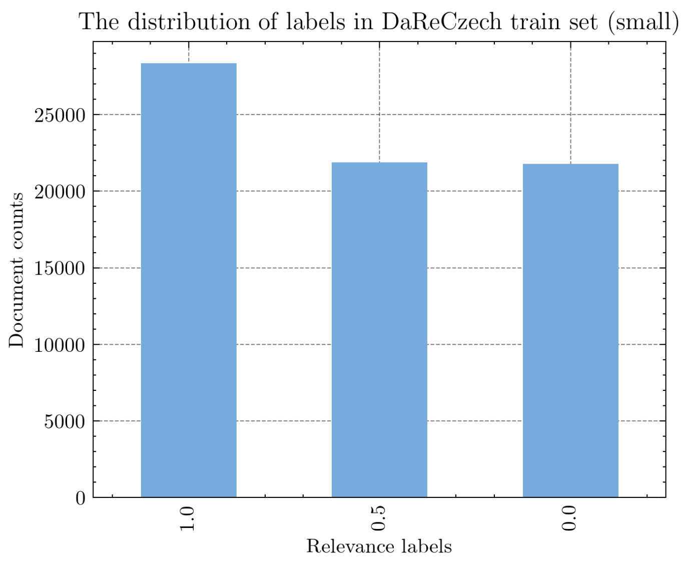
We create two versions of training data – a small and a big version. For the small version, we use the original *train-small* dataset. The bigger dataset is created by subsampling the original *train-big* data; we filter queries that have at least 10 annotated examples, resulting in dataset of more than 24K queries and 752K records. Table 4.3 shows the total number of records after filtering in each individual data partition. The test set is also cleared of empty documents and duplicates. The label distribution for the test set is a bit different than for the training set, as can be seen in Figure 4.1. Similarly as authors of the dataset, we consider the document in the test to be relevant if its label > 0.5. We fully remove test queries that do not have at least one relevant document (the queries that only have documents labelled as 0 or 0.25), as for these queries the Precision@10 metric that is computed for this dataset would be always zero.

Feature extraction

For feature extraction, we employ similar set of features as for the MS MARCO dataset, but in addition to the features that are extracted per entire document,

Index	Name	Feature description
1	Query length	The number of terms in the query.
2	Title length	The number of terms in the document title.
3	Body length	The number of terms in the document body.
4	Document length	The number of terms in the whole document.
5–7	Covered query number	The number of query terms covered by the document title, body and the whole document.
8–10	Covered query ratio	Covered query number divided by the total number of terms in the query.
11–25	TF	Sum, min, max, mean and median of term frequencies for each query term.
26-40	Normalized TF	Sum, min, max, mean and median of normalized term frequencies for each query term.
41-50	IDF	Sum, min, max, mean and median of inverse document frequency for each document term.
51-66	TF-IDF	Sum, min, max, mean and median of TF-IDF score for each query term.
67-69	BM25	BM25 score for the whole query.
70-72	LM.DIR	Score from Language Model with Dirichlet smoothing, for the whole query.
73-75	LM.JM	Score from Language Model with Jelinek Mercer smoothing, for the whole query.
76	URL length	The number of characters in document URL.
77	URL slashes	The number of times slash occurs in the document URL.

Table 4.4: The full description of the features used for the DaReCzech dataset.



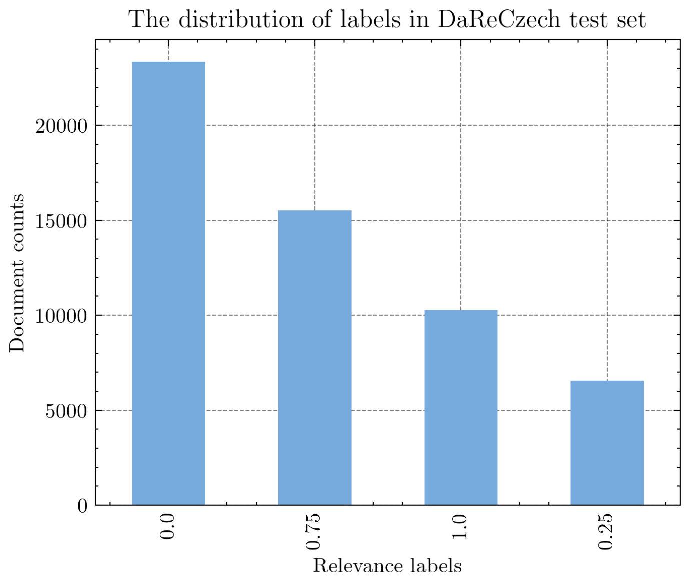


Figure 4.1: The label distribution for the DaReCzech train and test after removing empty documents.

we compute all features separately per *title* and *body* field. As nltk does not fully support Czech language, we use Elasticsearch Analyze API to extract terms from queries. Since the URLs for the documents are also available, we use them to add two new features – the length of URL and the number of slashes in it. This gives us 77 features in total; the full feature list is described in Table 4.4.

4.2.3 Semantic features

All the implemented features described in the previous section were created based on the document and query terms. Our hypothesis is that adding a semantic factor to our model may be able to overcome the problems of term-based features and improve the model performance. To capture the semantic meaning of queries and documents, we introduce an additional feature, which is derived from a pre-trained sentence transformer model.

MS MARCO

We use the *all-mpnet-base-v2* model from Hugging Face Transformers to compute the embeddings. This model was created by fine-tuning the MPNet [64] model introduced by Microsoft on a more than 1 billion sentence pairs. The model was trained to predict which sentence of randomly sampled sentences was paired with the given sentence in dataset. The concatenation from multiple datasets was used for this task, including 9,144,553 training tuples from MS MARCO. The model is intended to be used as a sentence or short paragraph encoder.

After tokenising the MS MARCO passages, we measure the average number of words in *large* train set, which is 66. The model can process an input of size of 384 words; sequences longer than that are truncated. The output of *all-mpnet-base-v2* is a 768-dimensional vector. We use *all-mpnet-base-v2* to compute the vector representation of each query and document pair separately and then compute the cosine similarity, which is added as an additional feature for the ranking model.

The reason for choosing this model is that as of April 2023, it is one of the most popular sentence models on Hugging Face (with more than 1,650,000 downloads for the previous month). Also, as mentioned, it was fine-tuned using MS MARCO passages and the input size is sufficient for this task.

DaReCzech

For Czech dataset, we use the *paraphrase-multilingual-mpnet-base-v2* model. Again, the output is a 768-dimensional vector. The embeddings are computed separately per document title and document body field. We compute the cosine similarity of query and both of the fields to obtain two new semantic features that are added to the data.

4.3 Semantic search

To experiment with improving the overall score of the implemented ranking system for MS MARCO, we carry out tests with semantic baseline. To create a semantic search system, we use Hierarchical Navigable Small World (HNSW) algorithm implemented in hnswlib [65]. HNSW is a graph-based approximate nearest neighbor search algorithm, and it is among the best-performing algorithms for vector similarity search [66]. HNSW consists of a multi-layer structure consisting of hierarchical set of proximity graphs for nested collections of the stored elements which is incrementally built.

Hnswlib is a C++ implementation of HNSW algorithm with Python interface. We compute vector representations for all 8.8M passages from MS MARCO dataset as described in the previous section and then create an index with cosine similarity set as a distance. The vectorization and index creation were the most computationally expensive operations in our ranking pipeline, the whole process took about 30 hours on GPU. When building the semantic index, we set the parameters M and $ef\_construction$. The parameter M controls the number of bi-directional links created for every new element during construction. The authors recommend the range of 2–100, where low M are functional for datasets with low dimensionality. For high dimensional embeddings (768 in our case), higher M is required (authors state the range 48–64, while the range 12–48 is desribed as reasonable for most the use cases). Therefore we set parameter M to 48. The $ef\_construction$ controls the ratio between index construction time and index accuracy. Higher values of this parameter lead to longer construction phase. We set $ef\_construction$ to 200.

4.4 Model selection and training

We choose LambdaMART as our LTR model based on the following facts. As described earlier, LambdaMART is considered one of the state-of-the-art traditional LTR models. Even though traditional LTR methods have become inferior in performance compared to deep neural approaches on some text-based datasets, the deep learning methods still have some drawbacks. The most significant difference between neural rankers and the traditional approaches are that deep learning rankers do not need hand-crafted features, as they are able to directly work with raw texts or word tokens. At first, this can be seen as a major advantage, on the other hand, as authors in [67] state, this makes the model more expensive at learning and query processing times. Also, researchers [23, 22] show, that when using hand-crafted features, traditional LTR models often outperform the neural rankers.

Generally, neural networks need large datasets to perform well, while models based on gradient boosted trees are known for their advantage in handling smaller data [68]. As the aim of this thesis is to experiment with size-limited data, we need a model that is less prone to overfitting with smaller datasets.

Furthermore, as researches in [22] point out, many publications that compare neural approaches to LambdaMART use the RankLib (or other inferior implementation), which was proven to substantially underperform the implementation from LightGBM library. It can be arguable whether some of these models are as effective as they claim to be.

4.4.1 Training

For training, we use the implementation of the LambdaMART algorithm from the LightGBM library. We find the optimal set of hyperparameters using optuna [69], an automatic hyperparameter tuning framework. For MS MARCO we employ cross-validation using *GroupKFold* with 5 folds from the scikit-learn [70] library. For DaReCzech we use fixed validation set we were provided with for tuning. We set the model's objective as *lambdarank* with *gbdt* boosting type, which gives LambdaMART algorithm. NDCG is set as the metric to optimise. The ranges and values for the hyperparameter tuning were set based on the recommendations in the official documentation[8] and the recommendations in [71]. LightGBM offers over 100 parameters that can be specified. The following summary explains the chosen parameters and our motivation to tune them; the full list of optimal parameters found for each model variation is attached in Table A.1 in Appendix A.

n_estimators We use up to 5000 estimators for our model, but with early stopping rounds set to 300, so the training is interrupted when the validation score stops improving.

learning_rate We experiment with setting learning rate, which controls the learning speed, to a value from the range $[0.01, 0.3]$.

num_leaves This attribute affects the tree shape as it controls the maximum number of leaves in one tree. We try the values from range $[8, 256]$.

max_depth Another tree shape controlling parameter is the maximum depth of a single tree. We experiment with not limiting the size of the tree as set by default for the model (value of -1), and also try to limit the maximum depth to a value in range $[3, 20]$. Smaller trees increase the training speed.

max_bin LightGBM buckets the continuous features into discrete bins. This is done in order to improve training speed and save memory during training. This parameter controls the number of bins the features are binned into. According to to the documentation, larger max_bin values increase accuracy, but slow down the training. We tune the hyperparameter to the value in range of $[255, 300]$.

min_data_in_leaf Minimum number of data points in one leaf affects the tree growth, as it specifies the condition under which a leaf will split. It can be used to avoid overfitting. It is recommended to use larger values for larger datasets, we experiment with range from 5 to 300.

[8]https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

min_gain_to_split Another tree growth contributing parameter is the minimum gain a leaf has to achieve in order to split. The default value is 0, meaning no limitations, we experiment with setting this value up to 15.

feature_fraction As some of the variables were influencing model's decision making process too strongly, we add the feature fraction parameter which limits the number of features that each iteration (tree) has available for training. We work with range $[0.5, 1]$.

bagging_fraction To force the model to generalise, we also experiment with bagging. At each iteration, the model selects a specified fraction of data points to use for training. Similarly as above, we tune the parameter based on the range $[0.5, 1]$.

Chapter **5**

Results

In the experiments, we aim to answer the research questions RQ1, RQ2 and RQ3 defined in the introduction of this thesis. This chapter focuses on presenting and discussing the achieved results.

5.1 RQ1 and RQ2

To determine whether LTR models improve the baseline IR system, we perform the following steps. As a baseline, we use a BM25 search system implemented in Elasticsearch with default parameters. We use the Elastic Ranking API to obtain the baseline performance measure for our test data. To evaluate other solutions, we use Elasticsearch as the first stage of the pipeline – we retrieve the top 100 documents for each query, hence the notation BM25_{100}. For each query and its 100 retrieved documents we extract the features and as a final step use the trained model to re-rank the result. The scoring function of the implemented LambdaMART algorithm is pointwise, meaning the output of the model is a relevance score for each query-document pair. The predicted scores do not have a fixed range, they can be negative and do not coincide with the relevance judgement labels provided to the model during training phase. The final ranked list of documents is constructed by sorting the output of the model by this score.

5.1.1 MS MARCO

As an evaluation metric for the MS MARCO dataset, we use the official leaderboard metric. We measure the mean reciprocal rank (MRR) at the top 10 and top 100 retrieved passages. Depending on the training data subset on which the model was trained, we use the notation LTR_{name}, where the data names correspond to those introduced in Table 4.1. The notation $+s$ means the model uses the train data with an additional semantic feature.

Solution	MRR@10	MRR@100
BM25 (Baseline)	0.18313	0.19437
BM25_{100} + LTR_{mini}	0.19597	0.2066
BM25_{100} + LTR_{small}	0.19747	0.20833
BM25_{100} + LTR_{medium}	0.20201	0.21269
BM25_{100} + LTR_{large}	**0.20374**	**0.21451**

Table 5.1: The best achieved MRR scores on MS MARCO.

Solution	MRR@10	MRR@100
BM25 (Baseline)	0.18313	0.19437
BM25 + semantic only	0.31787	0.32419
BM25_{100} + LTR_{mini+s}	0.32420	0.3299979
BM25_{100} + $\text{LTR}_{small+s}$	0.32731	0.33320
BM25_{100} + $\text{LTR}_{medium+s}$	0.32985	0.33562
BM25_{100} + $\text{LTR}_{large+s}$	**0.33031**	**0.33612**

Table 5.2: The best achieved MRR scores on MS MARCO with semantic feature.

Term-based features

The results in Table 5.1 show the mean reciprocal ranks for our models with a basic set of features (features in Table 4.2). We observe that, as expected, the performance of the model increases with the size of training samples. What is more, all implemented LTR solutions outperform the BM25 baseline. Even as little as 3K training queries were enough for the model to increase the MRR score by $\Delta = 0.012$.

Semantic features

Table 5.2 shows the results after adding the cosine similarity feature. It confirms our hypothesis that the ability to capture semantic meaning of sentences improves the model performance. Compared to the baseline, the improvement is by a large margin of $\Delta = 0.147$ for MRR@10 and $\Delta = 0.142$ for MRR@100 in the case of the *large* train set. Figure 5.1 shows how the feature importance changes after the cosine similarity of the query and the document is added – it becomes a strong predictor variable.

To validate the performance of our model, we conduct an additional experiment, where instead of using the LambdaMART model, we simply sort the results retrieved from Elasticsearch by cosine similarity of the query and document embedding. This setting is defined as *semantic only.* The LTR is still able to outperform this score, even when trained on the smallest dataset.

Solution	P@10	P@l
BM25 (Baseline)	0.25642	0.29314
BM25_{100} + LTR_{small}	0.28796	0.32128
BM25_{100} + LTR_{big}	**0.291198**	**0.38167**

Table 5.3: The best achieved P@10 scores on DaReCzech using BM25 as first-stage retrieval, measured on the whole set of 1,239,862 documents .

5.1.2 DaReCzech

To evaluate DaReCzech, we use the same performance measure as the data authors, Precision@10. After constructing the final sorted list of relevant documents, we measure the ratio of relevant documents (documents with the label > 0.5) in the top 10 retrieved documents. However, it is not clear how certain edge cases where handled from the original paper. For example, there are multiple test queries, that have less than 10 relevant documents. Some queries do not have relevant documents at all (they are paired only with documents labelled 0 or 0.25). If Precision@10 was computed without additional handling, such examples would automatically penalise the final score, even if all the available relevant documents for the query were found among the best 10 retrieved documents. Therefore, we compute two variants of precision. First, we compute a metric that does not specifically handle these queries and even if there are less than 10 relevant documents, it strictly computes the ratio of relevant documents among the top 10. Additionally, we compute Precision@l, where l denotes the total number of relevant documents for a given query if this number is less than 10, or 10 otherwise. We refer to these metrics as P@10 and P@l. However, it is worth noting that the computed scores are not fully comparable with the original DaReCzech paper, since in our test set the empty documents and all queries without relevant documents were removed. But our aim is to compare the improvement over the baseline, for which the score was calculated in the same way as stated above.

Term-based features

Table 5.3 shows the scores obtained for a term-based set of features. Figure 5.2 and Figure 5.3 show the feature importance for the final model (cropped to only the 10 most significant features).

It is worth noting that the *train-small* set is not a subset of *train-big*, in fact, these datasets are disjunctive. Therefore, the difference in scores could be boosted by better data samples that the model benefits from. Still, there is a notable improvement over the BM25 baseline in both variations of metrics for both versions of training data.

Re-ranker performance

In addition to the multi-stage ranking pipeline as used for MS MARCO, we conduct an experiment where we use the provided labelled test set. This experiment tests only the performance of the re-ranker, without the possible additional error created by the first-stage retrieval. After removing query-document pairs with empty documents and queries that do not have at least one relevant result, we sort the annotated documents provided for each query by the score predicted by the trained model. We also measure Precision@10 by simply sorting the documents by the BM25 score returned from Elasticsearch to serve as a baseline for this experiment. Table 5.4 demonstrates the result for this experiment.

Semantic features

Table 5.5 and 5.6 show the results after adding the features containing the cosine similarity between the query and the title and the cosine similarity between the query and the document body. While in the MS MARCO dataset, such feature resulted in a significant boost of performance, in case of DaReCzech, the models with semantic features improve the BM25 baseline but have very little performance gain compared to the model with term-based features only. Actually, the model trained on a small dataset performs worse than the LTR model using the basic set of features. There are several potential explanations for this result.

First, the DaReCzech dataset contains a lot of out-of-vocabulary terms, such as Czech names and brands. The sentence transformer models struggle with correctly interpreting such words, as they were trained on generic data, and on the contrary, the term-matching approaches benefit from it. Apart from that, the weaker performance could be caused by truncating the *body* field in the dataset to produce the vector representation. This field is much longer compared to the MS MARCO passage field, which could result in a loss of valuable information. Figure 5.4 supports this claim, since the cosine similarity for *title* field is one of the most important features, while the cosine similarity between the query and the *body* field does not influence the model as much. The other possibility is that the chosen model is not ideal for the Czech language.

5.2 RQ3

To answer the third research question, we conduct an experiment with the MS MARCO dataset, where instead of using BM25 provided by Elasticsearch as the first-stage retrieval, we use semantic search implemented using the HNSW algorithm as a baseline. The test queries are first vectorised using the same sentence transformer model as the train data. Similarly as for BM25, we re-

Solution	P@10	P@l
BM25 (Baseline)	0.46792	0.57634
LTR$_{small}$	0.51548	0.64145
LTR$_{big}$	**0.53892**	**0.70444**

Table 5.4: The best achieved P@10 scores on DaReCzech, applied directly on the test set of 5,247 query-document pairs, without using the first-stage retrieval.

Solution	P@10	P@l
BM25 (Baseline)	0.25642	0.29314
BM25$_{100}$ + LTR$_{small+s}$	0.28271	0.31302
BM25$_{100}$ + LTR$_{big+s}$	**0.301166**	**0.33123**

Table 5.5: The best achieved P@10 scores on DaReCzech using BM25 as first-stage retrieval and LTR model with semantic features.

Solution	P@10	P@l
BM25 (Baseline)	0.46792	0.57634
BM25$_{100}$ + LTR$_{small+s}$	0.47869	0.60052
BM25$_{100}$ + LTR$_{big+s}$	**0.55779**	**0.71009**

Table 5.6: The best achieved P@10 scores on DaReCzech, applied directly on the test set, without using the first-stage retrieval, and LTR model with semantic features.

trieve the most 100 relevant documents for each query embedding in the test set. This setting is called HNSW$_{100}$. First, we compute the mean reciprocal rank for the semantic search only, by using the order in which the documents were retrieved from the semantic index (sorted by cosine similarity). Then we re-rank the 100 retrieved documents and evaluate the performance for both versions of the LTR model – the one with term-based features and the one with additional semantic (cosine similarity of the query and the document) feature. This is done for all models for different data sizes. Table 5.7 shows the results obtained. It can be observed that models with only term-based features do not have the ability to improve the semantic baseline score, actually, they degrade the MRR scores by Δ up to 0.091. On the other hand, Table 5.8 shows that when using the model with the extra semantic feature, the score improves and the total ranking system performance is the best score obtained from all models, where MRR@10 equals to more than 0.35 for the largest dataset. The total document relevance quality was elevated by $\Delta = 0.02002$ in MRR@10 and $\Delta = 0.0247$ in MRR@100 compared to the best model using BM25 as the first-stage retrieval.

Solution	MRR@10	MRR@100
HNSW (Baseline)	**0.32904**	**0.34011**
HNSW_{100} + LTR_{mini}	0.23773	0.25225
HNSW_{100} + LTR_{small}	0.24196	0.25646
HNSW_{100} + LTR_{medium}	0.24045	0.25484
HNSW_{100} + LTR_{large}	0.24136	0.25646

Table 5.7: The best achieved MRR scores on MS MARCO using semantic index as a baseline.

Solution	MRR@10	MRR@100
HNSW (Baseline)	0.32904	0.34011
HNSW_{100} + LTR_{mini+s}	0.34210	0.35254
HNSW_{100} + $\text{LTR}_{small+s}$	0.34532	0.35603
HNSW_{100} + $\text{LTR}_{medium+s}$	0.3479	0.35871
HNSW_{100} + $\text{LTR}_{large+s}$	**0.35033**	**0.36079**

Table 5.8: The best achieved MRR scores on MS MARCO with semantic feature, using semantic index as a baseline.

5.3 Results summary

In the previous sections, we presented the results comparing the information retrieval baseline system with a system enhanced by Learning to Rank model. To sum it up, we propose the following answers for the defined research questions.

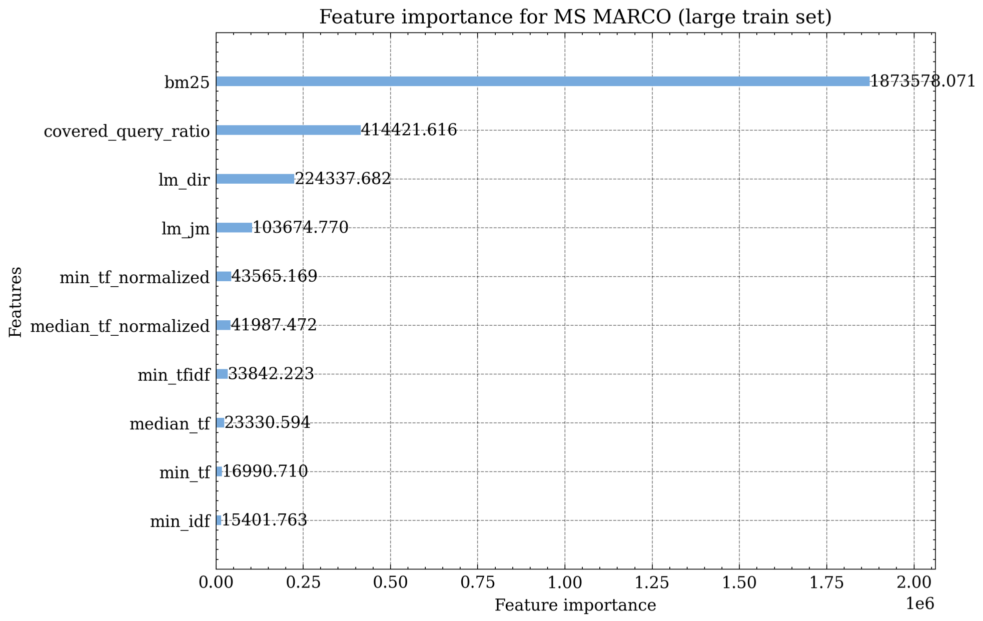
RQ1 For Research Question 1, we show that using a machine learning driven ranking model to sort documents retrieved from the baseline system has a positive effect on the result relevance. All implemented LTR solutions outperformed the BM25 baseline. Adding semantic feature in form of cosine similarity of the query and document embeddings improved the performance significantly for the MS MARCO dataset – the achieved performance gain is $\Delta = 0.147$ for MRR@10 and $\Delta = 0.142$ for MRR@100 for the *large* train set.

RQ2 For Research Question 2, our experiments with varying sizes of training data ranging from 3K to 77K queries from MS MARCO and 1K to 24K queries for DaReCzech showed that even the small versions of the training data were sufficient to improve the baseline results. The performance of the model increased with the size of the training samples; the best results were achieved by using the largest versions of the training data for both MS MARCO and DaReCzech. On the other hand, the variance between the quality of the smallest model compared to the model trained on more than 25× larger training set is not immense. Also, it should be pointed out, that even if refer to the datasets as *large*, their size is still relatively small compared to the original available training data. The achieved results confirm that Learning to Rank models are capable of delivering good performance even when working with limited data resources.

RQ3 The experiments related to Research Question 3 include replacing the term-based BM25 baseline with semantically driven search system implemented using the HNSW algorithm. We show that stronger baseline that uses the pre-trained sentence transformer outputs to find relevant document can still be improved by adding a re-ranking stage in the form of a traditional LTR model with term-based and transformer-based features. We also show that models trained on only term-based features were not enough to achieve a boost in performance, and, in contrast to models with a semantic similarity feature, they notably weaken the ranking pipeline performance. By combining the semantic baseline with a LTR model trained on the *large* training data, we achieve the best performance out of all models – MRR@10 of 0.35033 and MRR@100 of 0.36079.

5.4 Discussion

When we compare the best achieved score on the MS MARCO dataset of MRR@10 of 0.35033 to the winning place solution on the MS MARCO leaderboard, there is still a notable gap of approximately 0.1. It is important to point out, that the objective of this thesis was to quantify the performance improvement Learning to Rank methods can bring compared to a term-matching baseline system, not to achieve the best performance on the leaderboard. Apart from that, our solution uses only a small percent of the original training data and the re-ranking was not performed on 1000 candidates, but only 100, and due to memory reasons we do not use the full version of development data, only the 10 % of it.



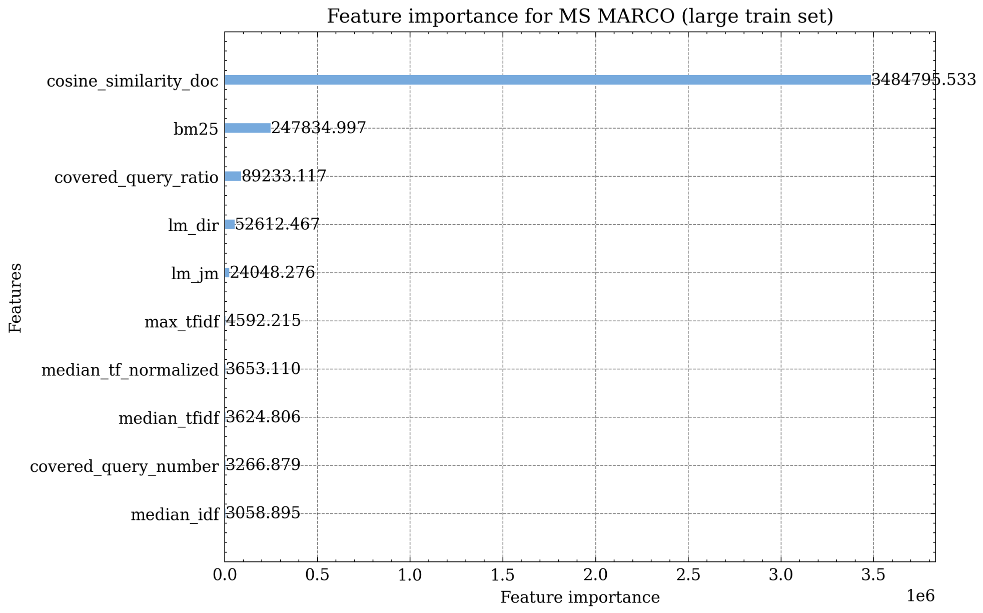


Figure 5.1: The comparision of feature importance for MS MARCO large train set using two sets of features.

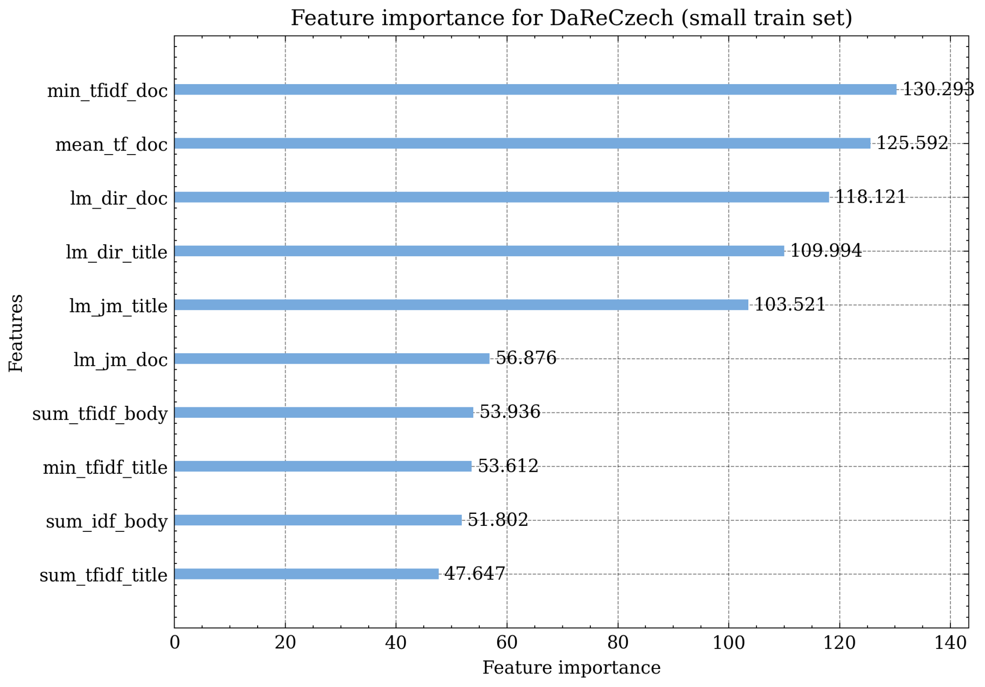


Figure 5.2: The feature importance for model trained on small DaReCzech dataset.

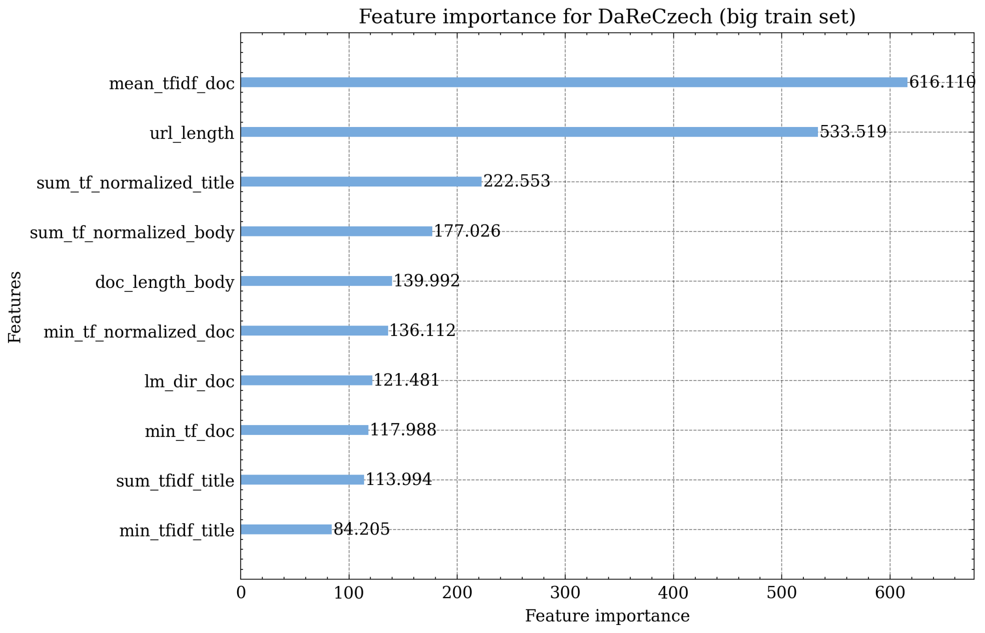


Figure 5.3: The feature importance for model trained on big DaReCzech dataset.

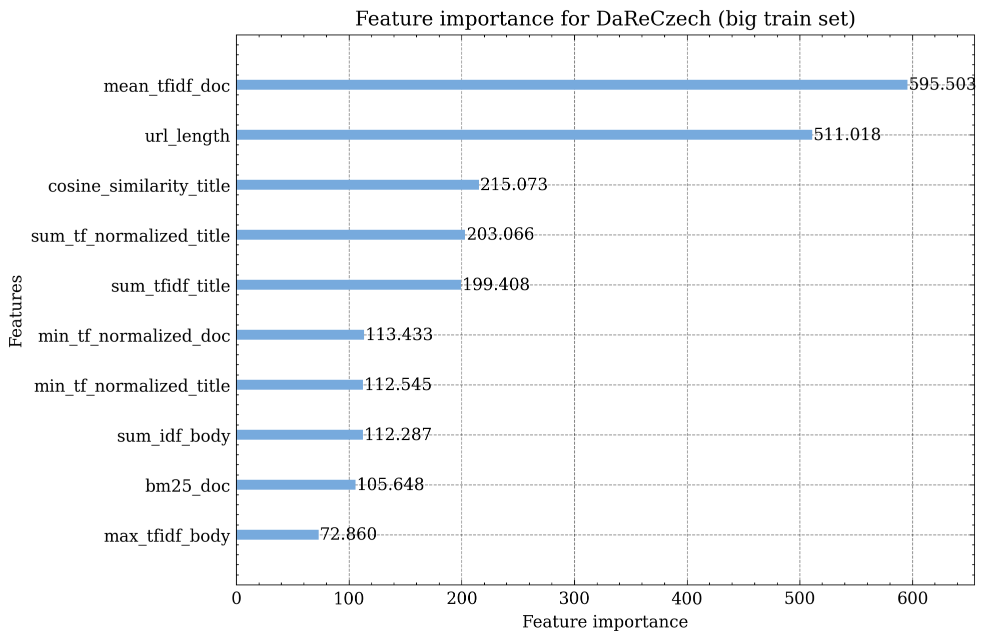


Figure 5.4: The feature importance for model trained on big DaReCzech dataset with semantic features.

Conclusion

In conclusion, the objective of this thesis was to address three research questions. The first research question focused on comparing a baseline IR system with an LTR-enabled system in order to determine the extent to which LTR models improve the result relevance. The second research question extended the first question by examining how much training data is needed for the LTR model to outperform the baseline system. Finally, the third research question explored the impact replacing the term-based first-stage retrieval with the pre-trained sentence transformer models has on the overall performance of the multi-stage ranking system.

To achieve the defined research goals, we provided an introduction to the basic concepts behind Information retrieval, traditional ranking methods, as well as a review of Learning to Rank. We reviewed the state-of-the-art ranking algorithms and analysed their advantages and limitations. We reviewed frequently used performance measures and provided an analysis of datasets suitable for this task. We provided a summary on the features that are typically used in this domain.

We conducted experiments on the MS MARCO and DaReCzech datasets, creating training data of varying sizes by subsampling the original datasets. We implemented a feature extraction module, that extracts 27 features for the MS MARCO dataset and 77 features for DaReCzech. We experimented with improving the quality of training data by adding a semantic feature represented as a cosine similarity of the query and document obtained from embedding representation from pre-trained sentence transformer model. For each created dataset, we trained a LambdaMART ranking model and found its optimal set of hyperparameters.

To evaluate the model performance, we implemented a multi-stage ranking system. We experimented with two types of first-stage retrieval models, namely a BM25 search system provided by Elasticsearch and a semantic search system implemented using hnswlib. We retrieved a pool of 100 candidate documents that were re-ranked by the created LambdaMART model. We compared the achieved scores to two types of baselines and we show that LTR models have to ability to improve the relevance of the final list of retrieved documents regardless the size of the traning data we used.

Overall, our findings demonstrate the effectiveness of traditional Learning to Rank methods in improving the ranking relevance when working with datasets of constrained sizes. In summary, this thesis provides valuable insight into the application of Learning to Rank methods on small datasets. The findings can be put to use in real-world scenarios where limited data availability is a common challenge.

Discussion

As a result, the practical implications of this thesis can be noteworthy to companies and inviduals that use traditional, term-based search systems based on TF-IDF or BM25 scores. First, the achieved results confirm, that adding a semantic factor can strikingly improve the search system performance. More importantly, the result relevance can be additionally boosted by Learning to Rank models. Our findings show, that organisations do not need annotated data in order of millions to enhance their search systems. As little as thousands queries are sufficient to successfully employ a supervised ranking model on top of a first-stage retrieval system. The labels for data can be collected from actual users of the systems, for example by adding feedback buttons to the existing search engine.

Future work

In the results presented in the previous chapter , we show that it is not the data size that notably influences the model performance, but rather the extracted features. As a future work, it is possible to experiment more with the feature extraction phase. One option is to experiment with extending the feature set by more complex features, such as translation-based or proximity based-features. It is also possible to test different sentence transformer models with different output embedding size for computing the transformer-based features.

For DaReCzech dataset, it may be interesting to try splitting the *body* field into multiple smaller chunks of sentences, that would be encoded separately by the sentence transformer. This could cover more information and the final cosine similarity feature could be either chosen as a maximum of cosine similarity of the query and all the parts of the field, or multiple features could be added in case of splitting the field into fixed sized parts.

Additionally, it may be possible to improve the overall MRR score achieved on MS MARCO by using a larger pool of documents retrieved by the first-stage retrieval system. In the original challenge, researchers typically work with a candidate set of 1000 passages. In this thesis we chose only 100 documents for memory and computational reasons – as there are approximately 7000 queries in the test set, it would require to retrieve, store, extract features and predict final scores for 7,000,000 passages in total.

Bibliography

[1] Singhal, A.; et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, volume 24, no. 4, 2001: pp. 35–43.

[2] Salton, G.; Wong, A.; et al. A vector space model for automatic indexing. *Communications of the ACM*, volume 18, no. 11, 1975: pp. 613–620.

[3] Deerwester, S.; Dumais, S. T.; et al. Indexing by latent semantic analysis. *Journal of the American society for information science*, volume 41, no. 6, 1990: pp. 391–407.

[4] He, B. *Probability Ranking Principle.* Boston, MA: Springer US, 2009, ISBN 978-0-387-39940-9, pp. 2168–2169, doi:10.1007/978-0-387-39940-9_930. Available from: `https://doi.org/10.1007/978-0-387-39940-9_930`

[5] Zhai, C.; Lafferty, J. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM SIGIR Forum*, volume 51, ACM New York, NY, USA, 2017, pp. 268–276.

[6] Sharipova, M. Language Models in Elasticsearch. 2018. Available from: `https://www.elastic.co/blog/language-models-in-elasticsearch`

[7] Voncina, K. *Learning to Rank-Feature engineering using a click model.* Dissertation thesis, 2018.

[8] Li, H. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, volume 94, no. 10, 2011: pp. 1854–1862.

[9] Crammer, K.; Singer, Y. Pranking with Ranking. In *Advances in Neural Information Processing Systems*, volume 14, edited by T. Dietterich; S. Becker; Z. Ghahramani, MIT Press, 2001. Available from: `https://proceedings.neurips.cc/paper/2001/file/5531a5834816222280f20d1ef9e95f69-Paper.pdf`

[10] Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001: pp. 1189–1232.

[11] Prasad, A. Regression Trees — Decision Tree for Regression — Machine Learning. August 2021. Available from: `https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047`

[12] Burges, C. From ranknet to lambdarank to lambdamart: An overview. *Learning*, volume 11, 01 2010.

[13] Freund, Y.; Iyer, R.; et al. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, volume 4, no. Nov, 2003: pp. 933–969.

[14] Burges, C.; Shaked, T.; et al. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 89–96.

[15] Freund, Y.; Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, volume 55, no. 1, 1997: pp. 119–139.

[16] Liu, T.-Y.; et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, volume 3, no. 3, 2009: pp. 225–331.

[17] Modrý, M. *Learning to Rank Algorithms.* Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2013, diploma thesis supervisor: Jan Šedivý.

[18] Burges, C.; Ragno, R.; et al. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, volume 19, 2006.

[19] Cao, Z.; Qin, T.; et al. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, New York, NY, USA: Association for Computing Machinery, 2007, ISBN 9781595937933, p. 129–136, doi: 10.1145/1273496.1273513. Available from: `https://doi.org/10.1145/1273496.1273513`

[20] Xia, F.; Liu, T.-Y.; et al. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1192–1199.

[21] Jennrich, R. I.; Sampson, P. Newton-Raphson and related algorithms for maximum likelihood variance component estimation. *Technometrics*, volume 18, no. 1, 1976: pp. 11–17.

[22] Qin, Z.; Yan, L.; et al. Are neural rankers still outperformed by gradient boosted decision trees? 2021.

[23] Dato, D.; MacAvaney, S.; et al. The Istella22 Dataset: Bridging Traditional and Neural Learning to Rank Evaluation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 3099–3107.

[24] Guo, J.; Fan, Y.; et al. A Deep Look into Neural Ranking Models for Information Retrieval. *CoRR*, volume abs/1903.06902, 2019, `1903.06902`. Available from: `http://arxiv.org/abs/1903.06902`

[25] Pang, L.; Lan, Y.; et al. DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval. *CoRR*, volume abs/1710.05649, 2017, `1710.05649`. Available from: `http://arxiv.org/abs/1710.05649`

[26] Fan, Y.; Guo, J.; et al. Modeling Diverse Relevance Patterns in Ad-hoc Retrieval. *CoRR*, volume abs/1805.05737, 2018, `1805.05737`. Available from: `http://arxiv.org/abs/1805.05737`

[27] Devlin, J.; Chang, M.; et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, volume abs/1810.04805, 2018, `1810.04805`. Available from: `http://arxiv.org/abs/1810.04805`

[28] Clark, K.; Luong, M.-T.; et al. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[29] Raffel, C.; Shazeer, N.; et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *CoRR*, volume abs/1910.10683, 2019, `1910.10683`. Available from: `http://arxiv.org/abs/1910.10683`

[30] Brown, T.; Mann, B.; et al. Language models are few-shot learners. *Advances in neural information processing systems*, volume 33, 2020: pp. 1877–1901.

[31] Liu, Y.; Ott, M.; et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, volume abs/1907.11692, 2019, `1907.11692`. Available from: `http://arxiv.org/abs/1907.11692`

[32] Vaswani, A.; Shazeer, N.; et al. Attention is all you need. *Advances in neural information processing systems*, volume 30, 2017.

[33] Lin, J.; Nogueira, R. F.; et al. Pretrained Transformers for Text Ranking: BERT and Beyond. *CoRR*, volume abs/2010.06467, 2020, 2010.06467. Available from: `https://arxiv.org/abs/2010.06467`

[34] Bromley, J.; Guyon, I.; et al. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, volume 6, 1993.

[35] Tax, N.; Bockting, S.; et al. A cross-benchmark comparison of 87 learning to rank methods. *Information processing & management*, volume 51, no. 6, 2015: pp. 757–772.

[36] Cai, Y.; Fan, Y.; et al. Semantic Models for the First-stage Retrieval: A Comprehensive Review. *CoRR*, volume abs/2103.04831, 2021, 2103.04831. Available from: `https://arxiv.org/abs/2103.04831`

[37] Nogueira, R.; Yang, W.; et al. Multi-stage document ranking with BERT. *arXiv preprint arXiv:1910.14424*, 2019.

[38] Chen, R.-C.; Gallagher, L.; et al. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 445–454.

[39] Joachims, T.; Granka, L.; et al. Accurately interpreting clickthrough data as implicit feedback. In *Acm Sigir Forum*, volume 51, Acm New York, NY, USA, 2017, pp. 4–11.

[40] Hu, Z.; Wang, Y.; et al. Unbiased lambdamart: an unbiased pairwise learning-to-rank algorithm. In *The World Wide Web Conference*, 2019, pp. 2830–2836.

[41] Nguyen, T.; Rosenberg, M.; et al. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *CoRR*, volume abs/1611.09268, 2016, 1611.09268. Available from: `http://arxiv.org/abs/1611.09268`

[42] Nogueira, R. F.; Cho, K. Passage Re-ranking with BERT. *CoRR*, volume abs/1901.04085, 2019, 1901.04085. Available from: `http://arxiv.org/abs/1901.04085`

[43] Craswell, N.; Mitra, B.; et al. Ms marco: Benchmarking ranking models in the large-data regime. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 1566–1576.

[44] Zhang, Y.; Long, D.; et al. HLATR: enhance multi-stage text retrieval with hybrid list aware transformer reranking. *arXiv preprint arXiv:2205.10569*, 2022.

[45] Han, S.; Wang, X.; et al. Learning-to-Rank with BERT in TF-Ranking. *CoRR*, volume abs/2004.08476, 2020, `2004.08476`. Available from: `https://arxiv.org/abs/2004.08476`

[46] Kocián, M.; Náplava, J.; et al. Siamese BERT-Based Model for Web Search Relevance Ranking Evaluated on a New Czech Dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, no. 11, Jun. 2022: pp. 12369–12377, doi:10.1609/aaai.v36i11.21502. Available from: `https://ojs.aaai.org/index.php/AAAI/article/view/21502`

[47] Boteva, V.; Gholipour, D.; et al. A Full-Text Learning to Rank Dataset for Medical Information Retrieval.

[48] Chapelle, O.; Chang, Y. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, PMLR, 2011, pp. 1–24.

[49] Qin, T.; Liu, T.-Y.; et al. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, volume 13, 08 2010: pp. 346–374, doi:10.1007/s10791-009-9123-y.

[50] Qin, T.; Liu, T. Introducing LETOR 4.0 Datasets. *CoRR*, volume abs/1306.2597, 2013, `1306.2597`. Available from: `http://arxiv.org/abs/1306.2597`

[51] Han, X.; Lei, S. Feature selection and model comparison on microsoft learning-to-rank data sets. *arXiv preprint arXiv:1803.05127*, 2018.

[52] Hu, C. *Learning to Rank in the Age of Muppets*. Master's thesis, University of Waterloo, 2022.

[53] Haveliwala, T.; et al. Efficient computation of PageRank. Technical report, Citeseer, 1999.

[54] Macdonald, C.; Santos, R. L.; et al. On the usefulness of query features for learning to rank. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 2559–2562.

[55] MacAvaney, S.; Yates, A.; et al. Simplified data wrangling with ir_datasets. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 2429–2436.

[56] Ke, G.; Meng, Q.; et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30, edited by I. Guyon; U. V. Luxburg; S. Bengio; H. Wallach; R. Fergus; S. Vishwanathan; R. Garnett, Curran Associates, Inc., 2017. Available from: `https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`

[57] Daoud, E. A. Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset.

[58] Wolf, T.; Debut, L.; et al. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. Available from: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`

[59] pandas development team, T. pandas-dev/pandas: Pandas. Nov. 2022, doi:10.5281/zenodo.7344967, if you use this software, please cite it as below. Available from: `https://doi.org/10.5281/zenodo.7344967`

[60] Harris, C. R.; Millman, K. J.; et al. Array programming with NumPy. *Nature*, volume 585, no. 7825, Sept. 2020: pp. 357–362, doi:10.1038/s41586-020-2649-2. Available from: `https://doi.org/10.1038/s41586-020-2649-2`

[61] Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, volume 9, no. 3, 2007: pp. 90–95, doi:10.1109/MCSE.2007.55.

[62] Garrett, J. D. garrettj403/SciencePlots. Sept. 2021, doi:10.5281/zenodo.4106649. Available from: `http://doi.org/10.5281/zenodo.4106649`

[63] Bird, S.; Klein, E.; et al. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[64] Song, K.; Tan, X.; et al. MPNet: Masked and Permuted Pre-training for Language Understanding. *arXiv preprint arXiv:2004.09297*, 2020.

[65] Malkov, Y. A.; Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, volume 42, no. 4, 2018: pp. 824–836.

[66] Bernhardsson, E. ANN Benchmarks. `https://github.com/erikbern/ann-benchmarks`, 2021.

[67] Silva, S. d. N.; et al. Applying machine learning to relevance evidence fusion at indexing time. 2020.

[68] Jiang, J.; Wang, R.; et al. Boosting tree-assisted multitask deep learning for small scientific datasets. *Journal of chemical information and modeling*, volume 60, no. 3, 2020: pp. 1235–1244.

[69] Akiba, T.; Sano, S.; et al. Optuna: A Next-generation Hyperparameter Optimization Framework. *CoRR*, volume abs/1907.10902, 2019, 1907.10902. Available from: http://arxiv.org/abs/1907.10902

[70] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[71] Tuychiev, B. Kaggler's Guide to LightGBM Hyperparameter Tuning with Optuna in 2021. Sep 2021. Available from: https://towardsdatascience.com/kagglers-guide-to-lightgbm-hyperparameter-tuning-with-optuna-in-2021-ed048d9838b5

Appendix A

Hyperparameters

Model	learning rate	num leaves	max depth	max bin	min gain to split	feature fraction	min data in leaf	bagg. fraction
MARCO$_{mini}$	0.01	172	-1	286	5.5	0.7	83	0.5
MARCO$_{small}$	0.3	256	4	260	11.2	0.4	20	1.0
MARCO$_{medium}$	0.1	102	-1	287	7.0	1.0	88	0.7
MARCO$_{large}$	0.05	16	14	283	1.5	0.9	20	1.0
MARCO$_{mini+s}$	0.05	128	6	266	6.8	0.9	20	1.0
MARCO$_{small+s}$	0.05	128	8	273	10.5	0.7	20	1.0
MARCO$_{medium+s}$	0.05	57	-1	269	13.0	0.9	51	0.5
MARCO$_{large+s}$	0.05	57	-1	269	13.0	0.9	51	0.5
DaReCzech$_{small}$	0.3	68	18	287	12.6	0.7	80	1.0
DaReCzech$_{big}$	0.3	68	18	287	12.6	0.7	80	1.0
DaReCzech$_{small+s}$	0.3	72	-1	292	14.0	0.9	142	0.6
DaReCzech$_{big+s}$	0.3	68	18	287	12.6	0.7	80	1.0

Table A.1: The set of hyperparameters used for singular models. The $+s$ denotes the usage of semantic features.

Appendix B

Acronyms

ANN Approximate Nearest Neighbor

BERT Bidirectional Encoder Representations from Transformers

GBRT Gradient Boosted Regression Trees

HNSW Hierarchical Navigable Small World

IDF Inverse Document Frequency

IR Information Retrieval

LMIR Language Model for Information Retrieval

LTR Learning to Rank

MAP Mean Average Precision

MRR Mean Reciprocal Rank

NDCG Normalized Discounted Cumulative Gain

NLP Natural Language Processing

SVD Singular Value Decomposition

TF Term Frequency

Appendix C

Contents of enclosed media

```
 README.MD ............................ the guide for the project set up
 learning__to_rank_thesis ............ Python package with source files
   configs ....... the configuration files for reproducing the experiments
   templates ........ the templates used for Elasticsearch Ranking API
 data .............................................. created data files
   msmarco ...................... train and test data for MS MARCO
   dareczech .................. train, val and test data for DaReCzech
 text ............................................. text-related files
   DP_Majtanova_Adriana_2023.pdf ...... a PDF rendering of the thesis
   majtaadr-assignment.pdf . a PDF rendering of the thesis assignment
     src ........................... the source files for the thesis text
 starter_guide.ipynb ................. starter guide jupyter notebook
```