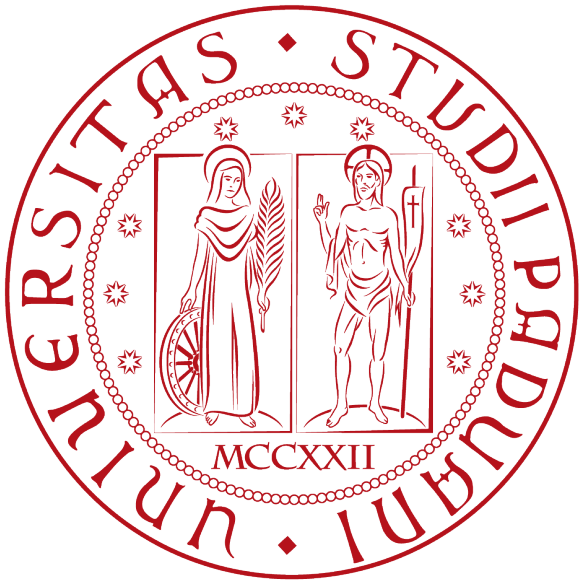
Deep Neural Models for Documents Retrieval and Ranking



Alberto Purpura

Supervisor: Prof. Gianmaria Silvello

Co-Supervisor: Prof. Gian Antonio Susto

Department of Information Engineering
University of Padua

This dissertation is submitted for the degree of
Doctor of Philosophy

November 2021

Abstract

Deep neural models revolutionized the research landscape in the *Information Retrieval (IR)* domain. Nowadays, they are employed at different stages in retrieval and ranking pipelines. For example, deep learning systems can automatically convert textual information into numerical features so that it can be later evaluated and compared by ranking models. Deep learning systems can also be effective in ranking items that have already been encoded into numerical features, i.e. comparing and ordering their representations according to a certain criterion.

This rapid adoption of deep learning models in ranking pipelines however was conducted considering them as as black-box systems, without a proportionate understanding of their key components. For this reason, in this thesis we aim at filling this gap in understanding and assessing the importance of each the the building blocks of neural IR models and to later improve their performance in different application scenarios, from text-matching to ranking.

Our contributions include an evaluation of the impact of probabilistic text representations in the text-matching task, the proposal of a new family of training paradigms for ranking models based on probability distributions, and a solution to improve the efficiency of neural reranking systems.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Ranking is a problem we encounter in a number of tasks we perform every day: from searching the Web to online shopping. The systems we use are based on a number of technologies developed over the past 70 years after the first computer scientists theorized the concept of an automated way to retrieve information.

Early *Information Retrieval (IR)* models performed simple boolean matching operations between keywords while modern ones rely on more complex statistical models and machine learning strategies to perform retrieval on textual content or data in other formats such as audio, image or video.

An IR system generally includes two components: an *information representation* and a *relevance estimation* model.

The first component has the role of representing information contained in a collection of items in a way that is practical for a machine to process. An implementation of such a system is for example a frequency-based term representation model such as *Term Frequency-Inverse Document Frequency (TF-IDF)* [3] – a representation model for textual data which leverages frequencies of terms contained in a textual document from a collection and in the whole documents set to represent them. This model was one of the first proposed solutions to represent textual information in a machine understandable format suitable for retrieval. The relevance estimation model on the other hand is a component dedicated to the estimation of the relevance of some content with respect to a user information need and can take different forms of arbitrary complexity, from a linear model to a deep neural network. In IR, relevance is an indicator of how strongly an item answers a user's information need. This can be expressed in different ways, i.e. through a textual query in the *ad-hoc* retrieval context.[1]

[1] Defined as the "standard retrieval task in which the user specifies his information need through a query which initiates a search (executed by the information system) for documents which are likely to be relevant to the user" [4].

A user's information need however is not only dependent on the information s/he directly submits to a search engine, but also on his/her personal characteristics – i.e. his/her age, location, cultural background etc. – and on outside factors – i.e. the time of the day the search query is issued.

Over the years, the development of new information representation models for textual documents and queries – from Latent Semantic Indexing [5] to Word2Vec [6] and, most recently, Transformer models [7] – encouraged a shift in technology also for relevance estimation strategies, i.e. from lexical matching models [4] to semantic ones [8], where the latter rely less on representations of words that only allow exact matching between terms – as in the TF-IDF model – and more on vector representations related to their semantic meaning.

This, in turn, encouraged the adoption of neural relevance estimation models [9] as a suitable modeling strategy to deal with new and richer information representation solutions. Indeed, neural and machine learning models in general offer a number of advantages compared to other strategies: (i) they provide a richer modeling solution compared to previous lexical relevance estimation models, and (ii) they offer an automated way to tune these parameters through machine learning optimization strategies.

Depending on the input information representation model used, machine learning and deep learning models applied to IR tasks can be distinguished between *LEarning TO Rank (LETOR)* and *Neural Information Retrieval (NIR)* models. In the first class, we group all machine learning – including deep learning – approaches, relying on manually-engineered features to represent their inputs. In the second group instead, we gather all models which automatically extract – relying on deep learning techniques – useful features to estimate the relevance of an item directly from its textual content, comparing it to the textual data in a user query.

As shown by the large number of NIR research works published over the past few years [10–12, 8, 13–18], the potential of these models attracted lots of interest from the IR community. Over the past decade, NIR models have been proven to be effective in learning abstract representations from raw input data and for tackling complex tasks otherwise out of reach for standard machine learning models. Most recently, deep learning models have also shown promising results in the LETOR domain [19–23]. This progress was also achieved thanks to the recent technological advancements and investments from the industry on the development of powerful hardware at a more accessible price.

Despite the large number of research papers published on the aforementioned topics, being deep learning a relatively new modeling strategy – especially in the IR field – the inner workings of neural models when applied to IR tasks are far from being fully understood.

Furthermore, as also shown by recent research papers [11], neural models have lots of potential for further performance improvements.

1.1 Research Questions

As pointed out in [24], the enthusiasm for NIR models sometimes contrasts the concerns regarding their actual reproducibility and effectiveness. Often, newly proposed NIR models provide only limited performance improvements over existing baselines and only in very few and controlled application scenarios. Moreover, when these models are presented and evaluated, crucial details on the specific experimental setup the authors employed – such as collection pre-processing strategies – are often overlooked. Indeed, a NIR model is a complex combination of different components that make its reproducibility more challenging than other IR systems. To prevent the usage of these models as black-box components in a search pipeline, we investigated on the following research question.

> **RQ1:** How can we improve the understanding of NIR models and facilitate the reproducibility of experiments involving them?

Following our in-depth analysis of NIR models, we shift our attention towards improving their performance. In particular, we focus on WE models, which are a crucial component for the performance of a NIR system. A WE model is the representation strategy NIR models rely on to interpret the textual information contained in documents and user queries. These models associate to each term – or sometimes also to each character or character n-gram – a real-valued vector which encodes its semantic meaning and relations to other words. Within this context, different approaches were proposed by the *Natural Language Processing (NLP)* community over the years, starting from Word2Vec [6], to fastText [25] and later *Embeddings from Language MOdels (ELMO)* [26] and *Bidirectional Encoder Representations from Transformers (BERT)* [27]. All these models were quickly adopted soon after their publication by the IR community, yielding important performance improvements across different NIR systems [28].

Another promising new line of research in this direction from the NLP domain proposes to employ *probabilistic models* to represent the semantic meaning of words. The intuition behind this modeling strategy is that words can be better represented by a probability distribution rather than just a real-valued vector identifying a point in the same multidimensional space. The main advantage claimed by this new family of models is the ability to explicitly encode all the different meanings of a word as a single probability distribution. This approach goes in a different direction than current state-of-the-art contextual word representation models based

on Transformers [29], where the context where a term is used helps shaping its representation at inference time with the help of very large – and not as efficient – deep learning models. Being a less popular approach, likely due to the lack of pre-existing software libraries to facilitate their integration in deep learning systems, probabilistic word representations have not yet been considered by the IR research community as an alternative to the aforementioned models. This motivated us to look into the following research problem.

> **RQ2:** Can probabilistic word representations improve the performance of NIR systems?

Another important limitation of NIR models is their computational cost, which is a direct consequence of interacting with complex word representation models. This issue however can be circumvented if we employ simpler – even though less general – representation strategies relying on manually-engineered features. This is the use case of LETOR models. These systems rely on a large number of simpler and faster-to-compute features that allow to improve the response time of the whole retrieval and reranking pipeline, without impacting its performance. Thanks to these properties, LETOR solutions are the most widespread approach employed in the industry when it comes to ranking problems. For these reasons, in the remainder of this thesis we focus on studying and improving this class of models.

In the LETOR domain, supervised machine learning approaches are the most popular choice. To train these models, a number of strategies have been proposed and implemented over the years. These all rely on similar input data, i.e. a set of query-document pairs, each associated to a relevance label.

However, despite being the most adopted solution, relying only on one label to represent the relevance of each query-document pair is not the only option. In fact, relevance is a subjective concept which is very complex to define, model and represent accurately. It does not only depend on the formulation of a user query but also on the search context and on the personal background of the user who is using the IR system. This is also confirmed by the relatively low inter-annotator agreement among expert judges in IR evaluation campaigns – often around 60-70% [30, 11] – compared to agreements achieved by annotators in the NLP domain which are often above 80% [11]. For this reason, with the goal of obtaining more reliable and less "noisy" relevance judgments, researchers from the IR community proposed different methods for the aggregation of relevance scores provided by annotators for the same query-document pair [31, 32]. Within this context, we explore a different approach to this problem and check if it is possible to train a LETOR model directly on relevance judgments distributions, preventing the loss of useful information when aggregating them into a single relevance label.

RQ3: Can we improve the performance of LETOR models by leveraging the entire information provided by relevance judgments distributions?

As mentioned earlier, a crucial limitation of neural models for IR is their efficiency. Indeed, when using a search engine, users expect very fast responses to their queries. This is the main reason why LETOR models are often preferred to text-based NIR ones in the industry domain. However, LETOR systems are not completely exempt from efficiency issues and this aspect becomes more relevant when neural models are employed. In the past few years, neural LETOR models achieved increasingly competitive performances [33, 34] when compared to traditional ones [22]. However, this was accomplished at the expense of larger and more complex architectures with a higher computational cost. For this reason, we investigate on potential solutions to improve the efficiency of a neural LETOR model so that it could be employed in a real-world reranking pipeline.

RQ4: How can we improve the efficiency of a neural LETOR model?

Overall, with our research work we aim at advancing the understanding of deep learning models applied to different IR tasks. At the same time, we aim to improve both the performance and efficiency of neural and traditional LETOR models.

1.2 Methods

To investigate the above-mentioned research problems, we consider: three popular NIR models such as *Deep Relevance Matching Model (DRMM)* [15], *Neural Vector Space Model (NVSM)* [13] and MatchPyramid [35]; a number of WE models including Elliptical *Probabilistic Word Embeddings (PWE)*; and different LETOR solutions.

RQ1: We focus on two paradigmatic NIR models, i.e. DRMM and NVSM. They are two deep learning solutions, one supervised and one unsupervised, to estimate the relevance of a textual document with respect to a user query. Through a reproducibility study, we analyze and evaluate them in different retrieval scenarios – i.e. experimental collections from different domains and in different languages – the role and impact of each of their core components and report the lessons we learned when seeking to replicate the performance results reported in their respective papers.

RQ2: We consider one of the best performing PWE strategies from the NLP domain, based on the Wasserstein Space of Elliptical Distributions [36]. We evaluate its performance in

the ad-hoc retrieval task, employing it as the word representation model in the MatchPyramid [35] NIR system. This experimental setup allows us to conduct a thorough analysis and comparison of PWE to other WE models such as Word2Vec, fastText and WordNet embeddings [37].

RQ3: We propose to improve the performance of LETOR models by leveraging on the full information provided by relevance judgments distributions. To accomplish this goal, we propose and evaluate a new family of probabilistic loss functions that are designed to use the information contained in raw relevance judgments distributions to train better performing LETOR models. We achieve this result bypassing the relevance labels aggregation step which is often performed when building a new experimental collection. We also conduct a crowdsourcing experiment creating a new LETOR collection to further validate our approach on real-world data. To evaluate the proposed functions we train both neural and traditional LETOR models such as LambdaMART [22].

RQ4: We propose and evaluate a new feature selection approach tailored for neural LETOR approaches. We believe feature selection approaches are a good fit for our research goal since they provide a cost-effective measure to reduce the complexity of an existing system without drastically changing its core components. Most approaches for feature selection in the LETOR literature however are based on simple correlation measures between input features and do not provide a solution tailored to any specific model needs [38]. Indeed, to estimate the relevance of an item, some neural models might learn to rely more on certain features following a specific distribution, while ignoring other ones which would be equally useful but are in a different range that is harder for the model to interpret. For this reason, we develop a novel approach for feature selection, specifically designed to improve the efficiency of neural LETOR models according to their own behavior.

1.3 Key Contributions

The main contributions of this thesis are summarized below.

RQ1: We provide an extensive evaluation of text-based NIR systems and their constituents through a reproducibility study of two paradigmatic models, i.e. DRMM and NVSM [24]. We analyze the impact of their core components such as the word representation strategy they use and highlight the most critical issues for their reproducibility, i.e. (i) the under-specification of the document pre-processing techniques adopted, (ii) the lack of references

to the exact external resources – e.g. the WE model – used to run the experiments, and (iii) the lack of a repository containing the implementation of the proposed approach. Finally, we provide an extensive evaluation of the considered NIR models performance on multiple retrieval collections ranging from the medical to the news domain and compare them to the most popular lexical alternatives. Our evaluation shows how the performance improvements promised by NIR models are often limited to a few controlled experimental scenarios and tightly linked to the experimental conditions.

RQ2: We explore a promising word representation strategy, i.e. PWE, and evaluate it in the context of ad-hoc retrieval using a popular NIR model, i.e. MatchPyramid [39]. Our experimental results [39] show promising performance improvements when using PWE to compute the similarity between query and document terms compared to traditional embedding models. However, they also reveal some efficiency limitations that hamper their applicability on large-scale search systems.

RQ3: We propose a new family of loss functions that allow the training of ranking models for IR relying on relevance judgments distributions. This training strategy allows models to benefit from the additional information available from the specific distribution of relevance judgments provided by different annotators for the same document-query pairs. The proposed loss functions show significant performance improvements when used to train both neural and traditional LETOR models.

RQ4: We propose a novel architecture-agnostic feature selection approach inspired by computer-vision techniques [38]. Our approach leverages on saliency maps to identify the most important features for relevance estimation used by a certain neural LETOR model. We evaluate our approach on two popular experimental collections selecting subsets of features of different sizes. The proposed approach could reduce the input size up to 20% of the total with minimal performance degradation and a substantial improvement in training and inference time.

1.4 Thesis Outline

This thesis is organized as follows. In **Chapter 2**, we provide an overview of the most popular NIR and LETOR pipelines. Next, we introduce their key components and highlight the advantages and limitations of different relevance estimation strategies. Finally, we introduce the evaluation measures and experimental collections that we employ throughout this thesis

to answer the aforementioned research questions. In **Chapter 3** we provide an evaluation of two popular NIR models, i.e. DRMM and NVSM, through a reproducibility study. We evaluate the models on a diverse set of retrieval domains and compare their performance to popular lexical and semantic retrieval approaches. Following the previous evaluation experiments aimed at understanding NIR reranking pipelines, we shift our attention towards new opportunities to improve their performance. We first focus on the evaluation of a new word representation paradigm with a strong potential, i.e. PWE. In **Chapter 4**, we assess the impact of PWE when used to represent queries and document terms on a popular NIR model, i.e. Matchpyramid. Next, we shift our focus towards LETOR models, which are a more efficient solution for document ranking compared to the NIR models evaluated so far. In **Chapter 5** we present a new family of probabilistic loss functions for ranking models and a novel neural LETOR architecture. In **Chapter 6** we propose an architecture-agnostic approach based on a neural LETOR model to reduce the size of its input limiting the impact on the system performance to a minimum. Finally, in **Chapter 7** we draw the conclusions of this thesis and discuss future research directions.

Chapter 2

Background

In this chapter, we present some background information on IR systems with a focus on solutions employing machine learning modeling strategies. We will begin with an introduction to the IR task and the main components of an information retrieval system, followed by a brief overview of the NIR and LETOR research areas. We conclude the chapter presenting the standard evaluation paradigm of an IR system and the evaluation measures we will employ throughout this work.

2.1 The Information Retrieval Task

Gerard Salton, one of the pioneers of IR, described it as: "a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information" [40]. Indeed, the goal of researchers in the IR community is to develop new strategies to make information accessible to a large number of people. Since its origin, the primary focus of the IR community has been on textual information and documents such as digitalized library archives and, most recently, Web pages and multimedia documents. Textual documents often have some amount of structure, i.e. we can recognize a title, an author or a date associated to it. However, most of the information they carry is in an unstructured format and presented in natural language form. This characteristic complicates the process of information retrieval compared to other search scenarios, i.e. on databases. Differently from a database, a document does not necessarily possess a machine-understandable structure that we can use to find the content we are looking for. Instead, we should rely on a number of techniques to extract and index this information in a way that is then efficiently processed in an automated way. Moreover, if in the early days of IR it would have been acceptable to rely on a trained professional as an intermediary to interact with a search engine, i.e. a librarian, nowadays we expect search engines to be easily accessible to everyone. For this reason,

modern search engines are expected to process queries that are in an unstructured format as they would be formulated by an untrained user. This is another differentiation factor from the databases world, where information is queried trough some structured language by trained professionals.

In general, search based on a user text query – also called *ad-hoc* search because the range of possible queries is huge and not prespecified [41] – is the most studied task in IR and will be the main focus in this thesis work.

When studying the problem of searching and retrieving information for a user, we need to also define the concept of *relevance*. Retrieving information that is relevant to a user translates in practice to providing some content that contains the information the user is looking for when s/he submitted a query to the search engine. This however is a very wide and abstract concept which is hard to frame into fixed categories. For instance, the relevance of an item not only relates to the concepts expressed in the user query but also to the location from which the request has been issued, the time of the day, or the personal background of the user receiving the item.

For this reason, retrieving documents by comparing their contents to the terms in a user query sometimes produces sub-optimal results in terms of relevance. One of the reasons why this strategy could be improved is that the same concept can be expressed using different natural language expressions that are not always the same across different query-document pairs. This issue is often referred to in IR as the *vocabulary mismatch problem*. Another important aspect to consider is the difference between *topical* and *user* relevance. In other words, a document which is relevant to a query might not be relevant for a certain user because of his/her personal characteristics, interests, location or background knowledge.

This motivated the IR community to implement different strategies to cope with the aforementioned problems that we will explore in the following sections.

2.2 Main Components of a Search Engine

At the highest level of abstraction, search engines support two functions: *index* and *search* [41]. The index function includes all actions required to build the structure that enables searching, i.e. *text acquisition*, *transformation* and *index creation*. The text acquisition module is in charge of identifying the set of documents that need to be indexed while the transformation module translates documents into features which will be stored in the index by the index creation module. Some actions performed by the latter module are for example: document parsing and tokenization, stopwords removal, stemming [41]. If we think about the structure of a Web search engine, the text acquisition module is represented by a crawling software,

scanning the Web for pages that need to be added to or updated in the index. Within the same example, the transformation module includes all functions required to parse HTML pages and to transform their contents into features to add to the index. Finally, the index module is represented by a specialized database that efficiently stores and queries the stored document features.

On the other hand, the search function is the component in charge of estimating the relevance of documents in a collection before presenting them to the final user of the system. A general representation of an IR search process is depicted in Figure 2.1.

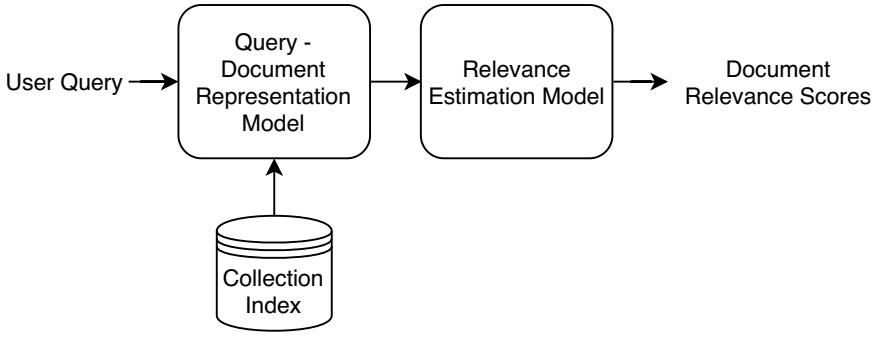


Fig. 2.1 Conceptual structure of an IR search process.

In this figure, we observe a first block which represents the operations an IR system performs to represent each query-document pair received in input. This is followed by another block which includes all the operations a model executes to estimate the relevance of each input item. The output relevance scores returned by the search component are often real values that are then used by the front-end module of a search engine to rank and present the documents in the searched collection in decreasing order of relevance.

Different solutions were proposed over the years for the implementation of both of the aforementioned representation and relevance estimation models. The earliest proposed solutions were boolean lexical retrieval models. These systems presented documents to the final user depending on whether they satisfied a certain boolean expression – no ranking function had been proposed yet to determine the order of the documents satisfying the query constraints. Later, other models were developed to also leverage the frequency distributions of words in a query-document pair to estimate its relevance. In the context of lexical retrieval approaches, the first block of Figure 2.1 consists in a series of operations to extract the frequency distributions of different terms in the input query and the documents of the collection from the previously computed index, while the second one implements a certain relevance estimation or ranking function based on this information. Later, with the popularization of machine learning approaches, new representation models for the content of

queries and documents were developed. These models – such as Word2Vec [6] – are able to represent a term through an embedded representation of its semantic meaning, expanding the ways terms from query-document pairs can be compared. These solutions encouraged the research of more advanced techniques for the representation of information contained in queries and documents. They also opened the way to the development of new approaches – i.e. NIR models – aimed at filling the semantic gap problem in IR by comparing different words based on the provided semantic representation. At the same time, a different family of machine learning approaches relying on manually engineered representations, i.e. LETOR models, grew in popularity. These models traditionally relied on a pool of features computed with efficient and easily scalable strategies that, when combined, provided a good overview of the relevance of a query-document pair. LETOR models also allowed to easily include in the representation of a query-document pair a number of features representing the retrieval context, independently from their textual content. This characteristic was helpful in bridging the differences between topical and user relevance we mentioned earlier. Overall, both NIR and LETOR approaches rely on machine learning – including deep learning – models to compute the relevance of a document with respect to a user query, and will both be discussed in this thesis work.

2.3 Lexical Retrieval Models

In general, lexical retrieval models estimate the relevance of a document relying on the frequency distributions of the terms – i.e. the lexicon – it shares with a user's query.

The first statistical approach to determine the relative importance of a term in a document was proposed by Peter Luhn [42] in the late 50s with the goal of performing document summarization. Later, IR researchers adopted and further developed Luhn's idea, applying it to the retrieval task, and proposing a large number of models of increasing complexity and performance. Overall, we can identify two main categories of models, i.e *vector space* and *probabilistic* ones.

In vector space models, a text is represented by a vector of terms. Once a predefined *vocabulary* is selected, each term in the vocabulary is associated to a dimension of a document vector. Thanks to this strategy we can represent textual documents through vectors of constant size. Since a document will likely contain a limited number of terms compared to the lexicon size, these vector representations will often be very sparse. To assess the *similarity* between a query and a document – considered as a proxy for their relevance – vector space models compare the respective document vectors through a similarity function. A typical similarity function is the cosine similarity which measures the cosine of the angle between two vectors

and provides a value between -1 and 1 where -1 indicates that the vectors are in opposite directions – and therefore very different – and 1 indicates a perfect overlap in their orientation – indicating a high similarity. In the context of vector space models, we can employ different strategies to associate an importance weight to each term in a document or query vector. One popular solution is to employ the TF-IDF term weighting strategy [43]. This approach assigns to each term a value equal to the product of its frequency in the considered document or query and its inverse document frequency in the collection – i.e. ratio between the total number of documents in the collection and the number of documents containing the given term.

Probabilistic models on the other hand are based on the principle that documents should be ranked according to their respective probability of being relevant to a given query. The goal of probabilistic models is to estimate this relevance probability. Different approaches were developed over the years, among which the most popular is likely the Okapi BM25 model [44]. According to this model, the relevance probability of a document d with regards to a query q is computed as a combination of the frequency of each query term in d and in the entire document corpus. This retrieval function also takes into account document lengths and can be adapted to better suit the collection statistical properties through some hyperparameters. Another successful probabilistic retrieval approach is the Query Likelihood Model (QLM) [45], where a document d is ranked according to the probability of generating a query q by randomly sampling the terms contained in it. This is formalized by the posterior probability $p(q|d)$ which can be estimated in its simplest form by:

$$p(t_q|d) = \frac{tf(t_q,d)}{|d|} \tag{2.1}$$

where $|d|$ indicates the total number of terms in d.

The BM25 and QLM models however consider by default only terms already employed by the user in his/her query formulation and ignore any proximity signals between their occurrence(s) in each document. To overcome these limitations, other models were proposed such as the translation model (TM) [46] – which proposes to estimate the relevance of a document d based on the likelihood of a query q of being generated by a "translation" process from the document:

$$p(t_q,d) = \sum_{t_d \in d} p(t_q|t_d)p(t_d,d) \tag{2.2}$$

– and the dependence model proposed in [47] – developed to take into account the relative positions of document terms. We observe that in these cases all terms in a document are taken into account for the estimation of its relevance. However, despite the performance

improvements achieved by these models, their relative complexity compared to previous solutions restricted their application mostly to the academic research community, while the industry settled for simpler and only slightly under-performing solutions such as the BM25 model [11].

Nonetheless, all the aforementioned models are limited by their lack of a solution to represent and compare the semantic meaning of different words – the so called vocabulary mismatch problem. This is often mitigated by: (i) computing correlations values between different words [46] and using these values to estimate their similarity; (ii) employing stemming techniques [48]; (iii) enriching queries or documents with more terms related to their respective content [49, 50]. A more reliable solution to this problem emerged in the early 2010s with the development of deep learning strategies and embedded representations of words [6] within the NLP community.

2.4 Neural Semantic Matching Models

With the rise in popularity of deep learning [9] and the development of embedded representations of words [6], the IR community started relying on new strategies to represent and compare the semantic meaning of different words. Indeed, embedded representations of words allow the concise description of the semantic meaning of a term and of its relations with others, through a real-valued vector in an n-dimensional space. The first and most popular of these strategies is Word2Vec [6], which relies on a shallow neural model to train – in an unsupervised way – an embedded representation of different words. These representations can then be used to estimate the semantic similarity between terms.

We can identify two different moments in the development of NIR models: (i) an *early stage* [51] where numerous semantic matching models were proposed, the majority of which relying on *Bag of Words (BOW)* representations of words such as Word2Vec [6]; and (ii) a *later stage* where the focus shifted towards the development of effective ways to leverage on Transformer-based contextual word representations [27, 28].

Among the early NIR approaches – following the analysis proposed in [8] – we can distinguish between *representation* or *interaction*-based models. Representation-based models are characterized by an asymmetric architecture which learns to produce, for each query-document pair it receives in input, separate representations which are then compared through a similarity function, e.g. cosine similarity. Examples from this group are Neural Vector Space Model (NVSM) [13] and Deep Structured Semantic Model (DSSM) [17]. Interaction-based models, on the other hand, rely on matching signals between query and document terms, derived from a certain WE model such as Word2Vec. These signals are then

analyzed by a neural model which learns to estimate the relevance of the corresponding items. Examples from this family of models are Deep Relevance Matching Model (DRMM) [15] and MatchPyramid [35].

Differently from the aforementioned approaches, later NIR systems – such as Contextualized Embeddings for Document Ranking (CEDR) [28], BIRCH [52] or the Deep Contextualized Term Weighting framework (DeepCT) [53] – rely on pre-trained Transformer models to compute the similarity between queries and documents.

The most popular Transformer-based solution for the contextual representation of words is BERT [27]. This model is generally employed after being pre-trained in an unsupervised setting and then fine-tuned depending on the target task. The main idea that inspired the development of BERT is to learn word representations by training a Transformer model to reconstruct a set of masked words in a text snippet. This strategy allowed the unsupervised training of a deep neural model and a set of WEs which can then be fine-tuned in a supervised way to satisfy the requirements of a different task such as text-matching.

Despite the sizeable improvement in the quality of word representations provided by this model, its efficiency limitations initially slowed down its adoption in the IR domain. Indeed, this approach requires to run the model every time we need to compute the representation of a term in an unseen context. A possible solution to improve the efficiency of BERT-based models is to train them to independently compute dense representations for queries and documents. In this way, document representations can be pre-computed offline, while query vectors are computed on-the-fly. This reduces the per-search computational cost and makes the model more scalable. An example of a model implementing this paradigm is the Dense Passage Retrieval (DPR) model, described in [54]. In DPR and other models following this paradigm, documents are retrieved and ranked relying on efficient k-Nearest Neighbor (k-NN) solutions such as FAISS [55] according to the similarity score between their dense representation and each query vector. The efficiency of these models however comes at the cost of compressing the information contained in documents and queries, hence limiting their accuracy when estimating the relevance of a document in a large collection. For this reason, at the time of writing, they are mostly employed in small-scale retrieval problems where there are only a few relevant items to rank and compare, as it is the case in the question answering domain [54].

Furthermore, due to efficiency constraints, most of the models mentioned in this section are employed as one of the final components of a reranking pipeline, receiving in input a small subset of documents retrieved with more efficient solutions such as lexical text matching models. Hence, due to the increased complexity of building and maintaining a retrieval and ranking pipeline relying on text-based deep learning models, the majority of the

solutions in the industry rely on simpler and more efficient machine learning solutions, i.e. LETOR models.

2.5 Learning to Rank Models

LETOR models have a relatively long history in IR [56, 57], reaching their popularity peak in the early 2010s [58, 11]. What differentiates these models the most from the aforementioned text or semantic matching approaches is their reliance on manually engineered features to represent query-document pairs. These are often represented by a feature vector dependent both on the user query characteristics, on different matching scores between the query and document text, and on other factors related to the document itself, i.e. the reputation of its creators. Query-document pairs representations are then fed to a machine learning model such as a random forest or a neural network [22] to estimate the relevance of the respective items. These models offer a competitive and efficient solution with respect to other lexical or semantic matching models, shifting the efforts from model development to feature engineering and training paradigms development. Indeed, most of the research in the LETOR domain traditionally focused on the development of new loss functions to train a ranking model. This was mainly motivated by the lack of a directly differentiable objective function that machine learning models could use to optimize the overall quality of a ranked list. In fact, there is still an open debate in the IR community on which is the most appropriate evaluation measure to assess the quality of a ranked list [59].

We can distinguish between three categories of LETOR models, depending on the training function they use, i.e. *pointwise*, *pairwise* and *listwise*.

Pointwise loss functions consider the relevance score returned by a LETOR model for a query-document pair and compute the model's error as done in regression problems, i.e. minimizing a loss function such as the *Mean Squared Error (MSE)* between the predicted and provided relevance score of each training data point. This class of losses can be formalized as:

$$\text{Pointwise}(q,d) = f(s(q,d),t), \tag{2.3}$$

where the function f can have different formulations and t indicates the true relevance score of each query-document pair (q,d).

Next, pairwise loss functions consider pairs of documents and compare their relevance scores one pair at a time. The goal of these class of functions is to minimize the pairwise ranking error of a model, i.e. to minimize the number of pairs of documents in a ranked list where a relevant one is ranked lower than a not-relevant one. This class of losses can be

formalized as:

$$\text{Pairwise}(q, d_i, d_j) = f(s(q, d_i), s(q, d_j)), \tag{2.4}$$

where the function f can have different formulations.

Finally, listwise loss functions differ from the previous ones because they estimate the quality of the predictions of a model for a given query comparing the relevance scores associated to multiple documents at once, i.e.

$$\text{Listwise}(q, d_i, ..., d_n) = f(s(q, d_i), ..., f(s, d_n)). \tag{2.5}$$

The most popular LETOR strategies are RankNet, LambdaRank and LambdaMART [22]. Where LambdaMART is the boosted tree version of LambdaRank [60], which is in turn based on the RankNet [61] model.

The main idea behind RankNet is to train a machine learning model – i.e., a Random Forest (RF) – to estimate the relevance of input documents with respect to a given query. The RankNet model is trained relying on a *pairwise* loss function based on the cross-entropy [9] loss function:

$$\text{Pairwise}(P_{ij}) = -\overline{P_{ij}} \log P_{ij} - (1 - \overline{P_{ij}}) \log(1 - P_{ij}), \tag{2.6}$$

where $P_{ij} = (1 + e^{-\sigma(s(q,d_i) - s(q,d_j))})^{-1}$ and $\overline{P_{ij}}$ is the expected difference between the relevance judgments associated to d_i and d_j.

Although RankNet was showing already good performance results [61], the model was not optimizing ranking evaluation measures such as *normalized Discounted Cumulated Gain (nDCG)* – which however is not differentiable – but was instead focusing on minimizing the number of pairwise ranking errors. To solve this shortcoming of RankNet, the LambdaRank model was proposed [60]. The idea behind this approach is to relax the requirement of a differentiable loss function to focus on improve the quality of the loss gradients. Hence, the main contribution of LambdaRank is to rescale the gradients of the RankNet model according to the nDCG difference given by swapping each pair of documents d_i and d_j considered in the pairwise loss computation. This helped the model getting closer to the goal of directly optimizing IR not differentiable evaluation metrics, and to improve the model performance.

Lastly, LambdaMART [22] combined the advantages of *Multiple Additive Regression Trees (MART)* models [62] with LambdaRank. MART models improve the performance of random forests using gradient boosting [63], a technique to train ensemble models.

Most recently – enabled by the availability of larger training datasets [64], new neural architectures [29] and training strategies [21] – deep learning LETOR models also started achieving competitive performance results compared to LambdaMART models.

2.6 IR Systems Evaluation

Traditionally, retrieval systems have been evaluated following the Cranfield paradigm – originally developed by Cyril Cleverdon in the 60s [65]. This paradigm was then adopted and improved over the years by different evaluation campaigns such as the *Text REtrieval Conference (TREC)* or the *Conference and Labs of the Evaluation Forum (CLEF)*. This evaluation strategy is based on a predefined and fixed collection of documents, information needs – also referred to as *topics* – and relevance judgments. The effectiveness of a retrieval system is then evaluated based on its ability to retrieve and rank the documents which were previously identified as relevant for each topic by a pool of judges [66].

Document collections – also referred to as *corpora* – can be of different sizes, containing from a few hundreds of thousands of documents such as in the TREC Robust04 collection [67] to millions of them in the MS-MARCO collection [68]. Documents in ad-hoc retrieval collections consist of natural language text and are mostly unstructured, often containing only a title and/or section headings if they are organized in paragraphs. Documents can also vary in length from just a few sentences to hundreds or thousands of them.

Information needs are also expressed in natural language text in the ad-hoc retrieval context. Ad-hoc TREC topics for example contain three fields: *title*, *description* and *narrative*. The title field contains a few keywords that synthetically describe the topic in a similar way as how it would be described when performing a Web search. This is the field which is used the most to perform retrieval experiments. The description field is a more verbose description of the topic and of the desiderata for the retrieved documents. Finally, the narrative field is an even more verbose description of the topic with additional information not contained in the previous two fields that could help to identify relevant items.

Relevance judgments are often collected relying on a pool of judges that assess a subset of the experimental collections where there is a high probability of finding relevant documents. Often, more than one judge is asked to provide an assessment on the relevance of the same topic-document pair. This relevance judgment can be binary or graded [69], i.e. taking integer values in an integer range such as $[0, 2]$ where 0 indicates a not-relevant pair, 1 a partially-relevant, and 2 a relevant one. Non-judged topic-document pairs are normally considered as not-relevant. Since more than one relevance judgment is often collected for a topic-document pair, these scores are then aggregated with different techniques such as majority voting or with more complex strategies [32].

2.6.1 Evaluation Metrics

Given a set of queries $q \in Q$, a collection of documents $d \in D$ and a ranked list R, containing a subsets of documents in D judged as relevant by a certain retrieval model for a query q, we can define the following evaluation measures for the evaluation of an IR model.

Precision. Precision is a set-based measure – i.e., a measure which does not take into account the relative order of the items in the considered set – which measures the fraction of documents in a ranked list R which are relevant. It is defined as follows:

$$\text{Precision}(R,q) = \frac{\sum_{d \in R} \text{rel}(q,d)}{|R|}, \tag{2.7}$$

where $\text{rel}(\cdot)$ is the indicator function which is 1 if a query-document pair is relevant and 0 otherwise; and the $|\cdot|$ operator is used to indicate the cardinality of a set. It takes values in the $[0,1]$ range, where 1 is associated to the best-case scenario where only relevant documents appear in R (and the opposite for 0). When graded relevance judgments are available, we consider as relevant – unless explicitly stated – items associated to a relevance grade greater than 0. Indeed, relevance grades are often positive integers between 0 and 2 or 0 and 4. Sometimes, negative values are used to indicate spam, i.e. documents which are undesirable and should not appear in ranked lists in any position. In our case, we consider these documents as not-relevant.

Often, the precision measure is computed considering only the top-k items in a ranked list. In these cases, we indicate the Precision measure as P@k. This is often done since for certain applications, in particular for reranking pipelines, the top part of a ranked list is what most end-users will see. Therefore, it is frequent to evaluate reranking models considering the top 1, 3, 5, 10 or 100 items in each ranked list.

The same evaluation practice also applies to the following measures.

Recall. Recall is another set-based measure which indicates the fraction of relevant documents identified as relevant in the collection that appear in a ranked list R. Formally, it is defined as:

$$\text{Recall}(R,q) = \frac{\sum_{d \in R} \text{rel}(q,d)}{\sum_{d \in D} \text{rel}(q,d)}, \tag{2.8}$$

where we employ the same notation as above and relevance is considered as a binary value. It takes values in the $[0,1]$ range, where 1 is associated to the best-case scenario where only relevant documents appear in R (and the opposite for 0).

Average Precision. This measure computes the quality of a ranked list R by averaging the Precision measure at different cutoffs, where a relevant document appears in it. This is formalized as:

$$\mathrm{AP}(R,q) = \frac{\sum_{\mathrm{rank}_d, d \in R} \mathrm{Precision@rank}_d(R,q) * \mathrm{rel}(q,d)}{\sum_{d \in D} \mathrm{rel}(q,d)}, \tag{2.9}$$

where rank_d indicates the 1-based index of a document $d \in R$. This measure is often indicated as *Mean Average Precision (MAP)* when its value is averaged over all topics in an experimental collection. As for the previous evaluation measures, relevance is assumed to be a binary value. AP values also vary from 0 to 1 with the same interpretations of the aforementioned measures. Typically, the AP is not computed at a cutoff like the Precision or Recall measures. However, ranked lists are often cut after 1000 documents so this is also the maximum number of items that are often taken into account to compute the AP of a ranked list.

normalized Discounted Cumulated Gain (nDCG). The nDCG measure is the most popular evaluation method for ranked lists. Differently from Precision and Recall, this is not a set-based measure, i.e. its value depends on the order of documents in a ranked list R. It is also a metric which can take into account different relevance grades, i.e. non-binary values indicating the different degrees of relevance of an item. The nDCG measure is defined as follows:

$$\mathrm{nDCG}(R,q) = \frac{\mathrm{DCG(R,\, q)}}{\mathrm{IDCG}(R,q)}, \tag{2.10}$$

where IDCG represents the *Discounted Cumulated Gain (DCG)* of the "ideal" ranked list where documents are ranked in decreasing order of relevance. Thanks to this normalization strategy its values range from 0 to 1, where 1 is associated to the best-case scenario (and vice versa for 0). The DCG can be defined in different ways, the most popular one – and the one adopted in TREC evaluation campaigns – is the following:

$$\mathrm{DCG}(R,q) = \sum_{\mathrm{rank}_d,\ d \in R} \frac{\mathrm{rel}(q,d)}{\log_2(1+\mathrm{rank}_d)}, \tag{2.11}$$

where here the function $\mathrm{rel}(\cdot)$ returns the actual (also graded) relevance grade associated to the query-document pair (q,d) in the relevance judgments. Variants of this measure also compute the DCG value as:

$$\mathrm{DCG}(R,q) = \sum_{\mathrm{rank}_d, d \in R} \frac{2^{\mathrm{rel}(q,d)} - 1}{\log_2(1+\mathrm{rank}_d)}. \tag{2.12}$$

In our evaluation, we employ the formulation reported in Eq. 2.11. As we do for Precision and Recall, we often compute the nDCG measure at different thresholds. We indicate this as nDCG@k. This measure is particularly popular not only because it can handle graded relevance judgments, but also because it models – though in a simplified way – the behavior of a user when viewing the list of items returned by a search engine or IR system in general. Indeed, a typical user scrolling through a results list would normally pay more attention to documents at the top of the ranked list. Hence, documents ranked higher should contribute more to the evaluation of the quality of a ranked list. This is achieved through the scaling factor proportional to each document's rank at the denominator of the DCG formulation in Eq. 2.11 and 2.12.

Expected Reciprocal Rank (ERR). The ERR measure was originally proposed in [70] to evaluate the quality of ranked lists in Web search engines and is defined as:

$$ERR = \sum_{r=1}^{n} \frac{1}{r} Prob(\text{user stops at position } r), \tag{2.13}$$

where n is the number of documents in a ranked list R and $Prob(\text{user stops at position } r)$ is defined as:

$$Prob(\text{user stops at position } r) = \prod_{i=1}^{r-1} (1 - R_i) R_r, \tag{2.14}$$

where $R_i = \frac{2_i^y - 1}{2^{y_{max}}}$ is the probability of relevance of a document, y_i its relevance grade and y_{max} the maximum relevance grade of a document in the collection. The ERR measure provides a quantification of the user satisfaction with the ranked list by modeling the probability of his/her satisfaction when reading the documents up to a certain position. A high probability of a user stopping after the document at rank r means the documents s/he saw up to there satisfied hers/his information need.

2.7 Ad-Hoc retrieval Experimental Collections

We introduce in this section the main characteristics of the ad-hoc retrieval collections that we employ in the remainder of this work to evaluate NIR semantic matching models and pipelines. The details such as the number of documents, and queries in each collection are reported in Table 2.1. All the considered collections that we describe in this section are

available from download from the TREC website with the respective topics and relevance judgements. [1]

	CLEF-DE	CLEF-FA	CLEF-IT	OHSUMED	NY	Robust04	WSJ	WT2g
Document Count	223,132	166,774	157,558	348,566	1,855,658	528,155	173,252	247,491
Query Count	95	100	90	97	50	249	150	50

Table 2.1 Statistics of the ad-hoc retrieval experimental collections that we adopt in our experiments.

CLEF. The CLEF Initiative (Conference and Labs of the Evaluation Forum, formerly known as Cross-Language Evaluation Forum) is a self-organized body whose main mission is to promote research, innovation, and development of information access systems with an emphasis on multilingual and multimodal information with various levels of structure. Since 2000 the CLEF has played a leading role in stimulating investigation and research in a wide range of key areas in the information retrieval domain, becoming well-known in the international IR community. [2] In this group, we distinguish between three document corpora, collected within the CLEF initiative, in different languages – i.e., CLEF-IT, CLEF-DE, CLEF-DA, respectively in Italian, German and Farsi – sharing the same genre and period [71, 72]. The Italian and German corpora are composed of newspaper articles from 1994 to 1995 and the Persian corpus (i.e., Farsi) from 1996 to 2002. The German and Italian news agency dispatches are all gathered from the Swiss news agency and comprise of the same corpus translated in different languages. Relevance labels in these collections are binary values where 0 indicates a not-relevant document and 1 a relevant one.

TREC New York Times (NY). The TREC New York Times (NY) collection was introduced in 2017 in the TREC 2017 Core Track [3] for news retrieval in the context of ad-hoc search. This collection consists of articles written and published by the *New York Times* between 1987 and 2007. Relevance labels in this collection are graded, with values in $\{0, 1, 2\}$ where 0 indicates a not-relevant document, 1 a partially-relevant one and 2 a relevant one.

[1] `https://trec.nist.gov`
[2] `http://www.clef-initiative.eu`
[3] `https://trec-core.github.io/2017/`

OHSUMED. This collection comprises of a set of 348,566 references/documents from MEDLINE, an on-line life sciences/biomedicine information database, consisting of titles and/or abstracts from many published medical journals. [4]

OHSUMED contains 106 topics, divided into 63 official topics and 43 pre-test topics that were rejected from official TREC-9 runs for a variety of reasons, often because they had too few relevance judgments. Topic fields are: title (patient description) and description (information need) [73]. Relevance labels in this collection are graded, with values in $\{0, 1, 2\}$ where 0 indicates a not-relevant document, 1 a partially-relevant one and 2 a relevant one.

Robust04. The Robust04 collection [74] is the most popular ad-hoc retrieval collection and a reference for most new ad-hoc retrieval models. This collection was proposed in 2004 within the TREC 2004 Robust track. [5] The goal of the Robust track is to improve the consistency of retrieval technology by focusing on poorly performing topics. An ad hoc task in TREC investigates the performance of systems that search a static set of documents using previously-unseen topics.

The document collection for the Robust track is the set of documents on both TREC Disks 4 and 5 minus the the Congressional Record on disk 4. [6] This document set includes material from the Financial Times Limited (1991, 1992, 1993, 1994), the Federal Register (1994), the Foreign Broadcast Information Service (1996), and the Los Angeles Times (1989, 1990). Relevance labels in this collection are graded, with values in $\{0, 1, 2\}$ where 0 indicates a not-relevant document, 1 a partially-relevant one and 2 a relevant one.

WSJ. The Wall Street Journal (WSJ) collection comprises articles published between 1986 and 1992 in the *Wall Street Journal*, available from the TREC TIPSTER collection [75] and associated to 150 different topics. Relevance labels in this collection are binary values where 0 indicates a not-relevant document and 1 a relevant one.

WT2g. The WT2g collection comprises about 250K documents crawled from the Web, collected and shared for the first time in the TREC 1999 Web Track [76]. Despite the availability of larger collections of Web documents such as the ClueWeb09 [77], we focus on the WT2g because of its more manageable size for our experimental evaluation.

Furthermore, since NIR models are often used in a reranking scenario, reranking a few hundreds to 1000 documents per topic retrieved with lexical models such as BM25, the performance difference between different neural approaches on these datasets would likely

[4] `https://www.nlm.nih.gov/bsd/medline.html`.
[5] `https://trec.nist.gov/data/robust/04.guidelines.html`
[6] `https://trec.nist.gov/data/cd45/index.html`

be more impacted by the retrieval approach employed or the available topics rather than the NIR model we aim to evaluate. Relevance labels in this collection are binary values where 0 indicates a not-relevant document and 1 a relevant one.

2.8 Learning to Rank Experimental Collections

The experimental collections that we consider in our experiments with LETOR models are the: MQ2007, MQ2008, MSLR-WEB30K [78] and OHSUMED [79], which are most frequently used experimental collections in the LETOR domain. All datasets are already organized in five different folds with the respective training, test and validation subsets.

MQ2007 and MQ2008. The MQ2007 and MQ2008 collections are based on the TREC Million Query Track [80] and contain $1,700$ and 800 queries, with a respective average of 40 and 18 assessed documents per query. Their documents are represented by 46 features containing the relevance scores associated to each document-topic pair by ranking models on different parts of the documents [78, 81]. The relevance scores used in these collection are integer values ranging from 0 to 2 indicating an increasing order of relevance.

MSLR-WEB30K. The MSLR-WEB30K is a subset of $30,000$ queries from the retired training set of the commercial search engine Microsoft Bing with an average number of 125 judged documents per query. Documents are represented by 136 features computed similarly as for the MQ2007 and MQ2008.

Differently to the MQ2007 and MQ2008 collections, here document features are not normalized. The relevance labels employed here are integers from 0 to 4 where 4 indicates a document with the highest relevance grade and 0 a not-relevant one.

OHSUMED. The OHSUMED [79] collection contains about 16K documents from MEDLINE and 106 queries with an average of 125 assessed documents. Documents are represented by 45 features and their labels are in the $\{0,1,2\}$ set indicating a not-relevant, partially-relevant and a relevant document, respectively.

Chapter 3

An Analysis of Neural Information Retrieval Models through a Reproducibility Study

3.1 Introduction

As mentioned in the previous chapters, over the past few years NIR models changed the research landscape of IR, attracting a great deal of attention from the research community. A dedicated workshop series was held at the *ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR)* [82, 83], an in depth monograph [84] and a special issue in the *Information Retrieval Journal* (IRJ) [85] have been published in 2018. Moreover, at major IR conferences like SIGIR, the number of papers employing deep learning is increasing at a fast pace – i.e., from one article published in 2015 to eleven articles published in 2017 [86].

Nevertheless, this burst of enthusiasm contrasts the concerns about the actual effectiveness of NIR methods, the understanding of their inner workings [87], and their reproducibility. Reproducibility is intended here as the possibility to replicate, more or less accurately, the results reported in a research paper by the authors of a new NIR system, relying on the provided description of the model in the same paper or on its source code, when available. However, we argue that reproducibility studies of NIR systems – beyond the advantage of exposing the limitations of the descriptions of NIR models and highlighting some good research practices – could also advance our understanding of their inner workings and limitations, and help avoiding their usage as black-box systems. For this reason, in this chapter we present a reproducibility study of two paradigmatic NIR models as an opportunity

to improve the general understanding of how these models work and to produce some guidelines for the reproducibility of NIR models in general.

3.2 Research Question

An NIR system is an ecosystem of components and for this reason its reproducibility and the understanding of its inner workings are quite challenging, even when the source code is available. Such systems often include text processing methods, lexical ranking models, WEs, optimizers, query expansion methods, and other traditional IR and NLP components. These are used to feed a shallow or deep neural network which in turns computes a relevance score for the input query-document pair(s). In this complex ecosystem, every single component has a sizable impact on the final performance. However, the interactions between each element of a NIR system have not been accurately investigated yet. For instance, how documents are pre-processed has implications on the creation of term embeddings. These, in turn, affect the neural model optimizer and parameter selection. Even though this domino effect holds true for almost all advanced IR systems, it is particularly accentuated in NIR ones – where it is hard (or even impossible) to understand why we obtain a specific output and thus how to detect the component which may be not working correctly. To reproduce the results achieved by NIR models, each system component needs to be finely tuned. Describing a neural network architecture in detail or providing the source code is usually not sufficient to reproduce the system successfully. When generalizing the application of NIR models on collections from different domains or in different languages, the problem is even more significant, as it needs to adapt and optimize many components on a different setting: i.e. from English news to German news, Web pages or medical documents.

Hence, with the popularity of NIR models, the analysis of these approaches, especially through reproducibility studies, becomes crucial for their in-depth understanding. In fact, the more we understand single system components and their interactions, the more we can generalize the approach and successfully transfer its performances to different domains.

However, the complexity of NIR models is not the only obstacle for their reproducibility. In many cases, NIR approaches work and are tested only on big sets of interaction proprietary data [88–90] which may not be easily accessible or available at all. On the other hand, to reproduce an experiment we require the original dataset or a reasonable approximation of it. Luckily, there are also several NIR approaches working on shared TREC collections [91–93, 2], especially in an ad-hoc retrieval setting.

The above open issues concerning NIR models motivated us to investigate the following research question:

RQ1: How can we improve the understanding of NIR models and facilitate the reproducibility of experiments involving them?

3.3 Methods

To achieve the above goals, we proceed reproducing and thoroughly evaluating two paradigmatic NIR systems: DRMM and NVSM. DRMM [92] was presented at the *25th ACM International Conference on Information and Knowledge Management* (CIKM 2016). DRMM achieved competitive results on re-ranking tasks in ad-hoc retrieval and it is still one of the reference NIR approaches. NVSM was published in the *ACM Transactions of Information Systems* in 2018 [2] and evaluated on shared TREC test collections, yielding also competitive results in ad-hoc retrieval. NVSM is one of the very few completely unsupervised existing NIR models, therefore it has excellent potential for generalization since it does not need any interaction data nor labeled data.

We replicate the experimental results in the paper presenting DRMM, leveraging on the source code shared by the authors. On the other hand, we re-implement NVSM from scratch in Python, relying on widely-used and consolidated libraries like TensorFlow. [1] This choice enables a straightforward comparison of NVSM with many other NIR models available in public repositories. [2] We reproduce the results of the original paper on the test collections used by the authors not only for NVSM, but also for the main baselines considered.

The goal of this analysis is to check if we could reproduce the results achieved both with NVSM and the proposed baselines and to report here the most valuable lessons we learned.

We consider four different perspectives in the analysis of DRMM and NVSM. First of all, we perform an in-depth evaluation of DRMM and NVSM and compare them with the most widely adopted lexical IR models such as TF-IDF, BM25, Query Likelihood Model (QLM), Divergence from Randomness (DFR), and other basic semantic models based on Word2Vec. The goal is to evaluate the potential of NIR approaches compared to a few widely-used and not necessarily heavily tuned IR models. Understanding NIR strengths and weaknesses can enhance their integration into full-stack IR systems, which employ a variety of pre- and post-retrieval components such as query expansion and relevance feedback.

Secondly, we test DRMM and NVSM on new search domains for which they were not initially designed: (i) the multilingual domain, where we consider Italian, German and Farsi [71, 72] news document collections from CLEF; (ii) the medical domain, where we consider the OHSUMED collection [73] composed of references/documents from MEDLINE

[1] `https://www.tensorflow.org/`
[2] `https://github.com/NTMC-Community/MatchZoo`

(the online life sciences/biomedicine information database); (iii) the Web domain, where we consider a small Web collection, namely the TREC WT2g [76]. Since NVSM does not scale to large document collections due to memory and time constraints, we chose collections of the same order of magnitude as those adopted in the original paper.

Thirdly, we perform an analysis of the impact of different word vector representations (i.e., WE models) on DRMM – considering other embedding models such as FastText [25], Word2Vec [6] and the WEs computed by NVSM [2]. Regarding NVSM, we cannot perform the same analysis as the joint learning of word and document embeddings is an inherent part of the model.

Finally, we perform a topic-by-topic analysis and comparison of selected NIR systems to the well-known BM25 lexical retrieval model. We highlight the performance differences among systems, describing the topics where NIR approaches are performing better than lexical models, and vice versa. Our aim in this case is to understand whether the two approaches are orthogonal, if they share common features, when it is more convenient to use a lexical model or a neural model.

3.4 Contributions

Our contributions regarding the aforementioned research questions are summarized below.

- We first conduct a reproducibility study of DRMM and NVSM on the original experimental collections they were evaluated on, formulating a few useful guidelines for the reproducibility of NIR systems;
- Next, we compare DRMM and NVSM to widely-used lexical and semantic IR models, expanding the original evaluation setup of the models and allowing for a broader understanding of the effectiveness of NIR approaches compared to other existing solutions;
- We also evaluate the aforementioned NIR models in heterogeneous search domains, i.e. the medical, news and Web search domains. This allows us to assess the generalization power of the considered models to different search contexts;
- We then shift our focus on analyzing and evaluating the impact of one of the most relevant components of NIR models, i.e. their word representation module. In this case, we assess the impact of employing different WE models on DRMM. Our analysis contributes to improve the general understanding of the impact of this crucial component in modern NIR systems;

- Finally, we conduct a topic-by-topic evaluation of the selected approaches comparing them to the BM25 retrieval model, highlighting a few situations where neural models have an advantage over traditional lexical ones.

3.5 Outline

This chapter is organized as follows. In Section 3.6 we provide a brief overview of the most popular representation and interaction based NIR systems and summarize recent research efforts in the IR reproducibility domain. In Section 2.7 we report the details of the experimental collections we use in this chapter's experiments while in Section 3.7, we describe the experimental setup employed in our tests; in Sections 3.8 and 3.9, we describe the DRMM and NVSM models, along with the results of our reproducibility experiments; in Section 3.10, we report a comparison of the selected NIR systems with traditional lexical approaches and other state-of-the-art models; in Section 3.11, the robustness of DRMM and NVSM is evaluated over a set of collections in different languages and from distinct domains; Section 3.12 assesses the impact of different WE models on DRMM; in Section 3.13, we perform an in-depth topic-by-topic analysis of the characteristics of the selected NIR models; and finally, in Section 3.14, we present our final remarks on this chapter's work. The source code and the settings we used for this study are available at the following URL: `https://github.com/giansilv/NeuralIR`.

3.6 Related work

Over the past few years, the increased availability of data and the success of deep neural networks in the NLP field, has promoted the diffusion of neural models also in the IR domain. As already mentioned in Section 2.4, existing NIR approaches can be broadly classified into representation and interaction based models [12]. In this section, we will present in more detail the most significant and influential models from these two categories.

3.6.1 Representation Based NIR models

The aim of representation-based models is to learn how to represent a query and a document and then use this representation to estimate their similarity. Within this class, one of the first proposed neural models to represent textual contents in queries and documents was the *supervised Deep Structured Semantic Model (DSSM)* [17]. DSSM is trained by maximizing the conditional likelihood of clicked documents given a query using click-through data. The

model employs a neural model combined with word hashing – which allows it to scale up and handle large vocabularies which are common in large-scale Web search applications – to compute a representation of queries and documents and then computes the similarity between the obtained representations using the cosine similarity function. DSSM was not only one of the first proposed interaction based NIR models, but was also an inspiration to other IR researchers to develop different architectural variants of this system such as *Convolutional Latent Semantic Model (CLSM)* [94] and *LSTM Deep Structured Semantic Model (LSTM-DSSM)* [95].

The application of deep neural networks to the IR task however is not only limited to supervised models but also extends to *unsupervised* ones. Along this line of research, we can further distinguish between two classes of models: (i) approaches that incorporate features from neural language models such as Doc2Vec (PV-DBOW architecture) [96], *Generalized Language Model (GLM)* [97], or the approaches described in [91] and [98]; and (ii) methods that learn representations of words and documents from scratch and use them directly for retrieval [99–101, 2, 102]. Within (ii), the works presented in [100, 101] introduce an unsupervised end-to-end representation learning model for product and expert search, respectively. Finally, the NVSM model [2] – that we will further analyze through this chapter – extends [100, 101] to ad-hoc retrieval by incorporating a notion of term specificity in the learned word representations. An extension of NVSM that integrates text matching and product substitutability for product search is also presented in [102].

3.6.2 Interaction Based NIR Models

Differently from representation based approaches, interaction-based models tackle the problem of predicting the relevance score between a query and a document computing the interactions between the query and document terms. DRMM is one of these approaches and the first that outperforms similar previous existing techniques for text matching such as ARC-II [16], and MatchPyramid [14]. Other successful approaches which belong to this category are *Match-SRNN* [103], *HIerarchical Neural maTching model (HiNT)* [104], *Kernel based Neural Ranking Model (K-NRM)* [105], *Convolutional Kernel-based Neural Ranking Model (Conv-KNRM)* [106], *A Position-Aware Neural IR Model for Relevance Matching (PACRR)* [107] and *A Context-Aware Neural IR Model for Ad-hoc Retrieval (Co-PACRR)* [108]. The architecture of Match-SRNN models the interaction between two texts as a recursive process. This means that the interaction of two texts at each position can be considered as a combination of interactions between their prefixes and words at a given position. This approach is similar to the one employed in MatchPyramid [14], even though the former uses a a *Spatial Recurrent Neural Network (SRNN)* instead of a regular

Convolutional Neural Network (CNN). Differently from previous models, HiNT focuses on differentiating between the diverse relevance patterns in a document given a query. This means that a document may be completely or partially relevant to a query as long as it provides sufficient information for users needs. For this reason, HiNT allows relevance signals at different granularities to compete with each other for final relevance assessment through a hierarchy of matching layers. Next, K-NRM uses a translation matrix approach that models word-level similarities via WEs, a kernel-pooling technique that uses kernels to extract multi-level soft match features, and a learning-to-rank layer that combines those features into the final ranking score. K-NRM also inspired other approaches such as Conv-KNRM which is similar to the former but computes the similarity between word n-grams after computing their representation with a CNN over the WE layer. Finally, PACRR and its variant Co-PACRR compute the relevance score of a query-document pair from multiple word n-gram similarity matrices processed first with a CNN and then with a *Recurrent Neural Network (RNN)*. In Co-PACRR, Hui et al. propose to employ – as an extension of the model proposed in PACRR – a context vector to enrich the matching signals, and replace the RNN with a simpler *Feed-Forward Neural Network (FFNN)*.

3.6.3 Reproducibility of NIR models

Along with the growing importance of NIR models, their reproducibility and evaluation is also becoming a central topic in the IR community. As we mentioned earlier, Lin initially criticized the "neural hype" in a recent SIGIR Forum paper [87] and more recently Wei et al. [109] critically examined the advances in NIR regarding the reproducibility of the systems, the data on which they are tested and the improvements over robust and well-tuned baselines. Indeed, the issue of reproducibility of search algorithms regards the IR field as a whole, not the only NIR. Recently, reproducibility also acquired a more central role in the IR community, with dedicated workshops [110–112] which raised awareness and proposed a reproducibility model – i.e., PRIMAD [113]; with a specific track at the *European Conference on Information Retrieval (ECIR)* since 2015; and, with dedicated journal special issues [114, 115]. Reproducibility efforts focused on several core topics in IR ranging from reproducing baselines [116, 117] and core system components [48] to evaluation [118, 119] and advanced applications [120]. Some works have focused on reproducing neural architectures for question answering [121], reproducing and generalizing the linear transformation of WEs [122], or replicating neural search models on ad-hoc test collections [123]. However, there has been no specific effort towards the reproducibility, analysis and generalization of NIR systems.

3.7 Experimental setup

To evaluate the DRMM and NVSM model, we rely on eleven experimental collections in different languages and from various domains. The collections characteristics are described in Section 2.7 and reported in detail in Table 2.1. For the retrieval experiments, we consider three combinations of the topic fields: the *title* (T) field only, the *description* (D) field only, or both the *title* and *description* (TD) fields. We compare the selected NIR models to other well-known semantic and lexical approaches for retrieval. The semantic models include Word2Vec and *Latent Dirichlet Allocation (LDA)*, while the lexical ones refer to QLM, BM25, TF-IDF, and the *Poisson estimation for randomness using Laplace succession for normalisation (PL2)* model [124] (hereafter DFR). In particular, for Word2Vec we consider the approaches originally proposed in [99], which define a document representation as the weighted sum of its WEs. We consider the unweighted sum, referred to as Word2Vec (add), and the sum weighted by the term's self-information, [3] referred to as Word2Vec (si). Whereas, for QLM, we consider the approaches with Jelinek-Mercer smoothing, referred to as QLM (jm), and with Dirichlet smoothing, referred to as QLM (dir) [126].

Reproducibility experiments. In the reproducibility experiments of DRMM, we employ the Robust04 collection [74], which is one of the two collections where the model was evaluated in the reference paper [92].

In the reproducibility experiments of NVSM, we use the same set of 6 newswire article collections from TREC – see Table 2.1 – adopted in the reference paper [2]. Four of the collections considered herein are subsets of the TIPSTER corpus [74]: *Associated Press 88-89* (AP 88-89), *Financial Times* (FT), *LA Times* (LA) and *Wall Street Journal* (WSJ) [75]. The remaining two collections are the Robust04 collection – based on TIPSTER Disk 4&5 minus CR – and the *New York Times* (NY) collection – which consists of articles written and published by the *New York Times* between 1987 and 2007.

Comparison with other lexical and semantic models. In this set of experiments, we compare DRMM and NVSM to other lexical (i.e. BM25, TF-IDF, DFR, QLM) and semantic models (i.e. Word2Vec (si)) on the Robust04 and NY collections. The aim is to compare the performance of NIR models considered to lexical and semantic matching techniques and to state-of-the-art retrieval approaches. The performances of other competitive approaches as the BM25+RM3 model in Anserini [117] and the one presented in [127] using BERT [7] to

[3] Self-information is a term specificity measure similar to IDF [125]

perform retrieval, are also considered for reference. We also report the performances of the best TREC systems when available.

Collection-based evaluation. In this set of experiments, we evaluate the impact of the domain and/or language of different collections on the performance of selected NIR models. The DRMM and NVSM models are evaluated on five collections from different domains and languages described in Section 2.7: WT2g [76], OHSUMED [73], CLEF [71, 72]. The detailed characteristics of the collections are reported in Table 2.1. For reference, we also report – whenever possible – the performance of the best TREC and CLEF systems on the experimental collections relative to the evaluation measures we consider.

Embedding-based evaluation. In this set of experiments, we evaluate DRMM on the Robust04, NY, WT2g and OSHUMED collections using WEs generated with different techniques. Indeed, different word representations can have a sizeable impact on the performance of neural models. This is also shown in [28], where the authors evaluate BERT [7] and ELMo [26] representations for retrieval. We consider four different types of WEs:

W2V: Word2Vec embeddings trained on each collection used to perform retrieval with Gensim [128], following the instruction in the DRMM reference paper [92]. To train this model, we consider a context window size of 10, an embeddings size of 300, 10 negative samples, a subsampling of frequent words of 10^{-4}, and we discard all terms appearing less than 10 times. We empirically select 10 as the best number of training epochs for the model after evaluating the system performance with a model trained for 5, 15 and 20 epochs. We also decided to apply punctuation and stopwords removal (using the INQUERY [129] stoplist), stemming (Krovetz [130]) and to remove all terms containing one or more digits or shorter than three characters as a pre-processing step on the document corpus before training the model;

fastText: fastText [25] embeddings trained with Gensim on each collection used to perform retrieval. fastText is an extension of Word2Vec to compute word representations based on character n-grams. In this case, we consider an embeddings size of 300, a context window of size 10, a subsampling of frequent words of 10^{-4}, 10 negative samples, and we discard words appearing in the collection less than 10 times. We train this model for 10 epochs on each of the selected experimental collections;

NVSM WE: the set of WEs learned and used by the NVSM model. The NVSM retrieval model works by learning both word and document representations from scratch in an unsupervised fashion and then using them to perform retrieval. Details on the training

and retrieval processes are described in Section 3.9. Since NVSM embeddings are learned – like Word2Vec embeddings – based on the cosine similarity metric, we can use them in the DRMM model without any modifications. As done for the other embedding models, we train a new model on each of the considered collections;

W2V Google: Word2Vec embeddings trained by Google on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. [4]

Topic-based evaluation. In this set of experiments, we conduct a topic-by-topic analysis on the performance of the selected NIR models and the BM25 model. The aim is to perform an in-depth comparison between lexical and semantic retrieval models. We consider Robust04, NY, WT2g, OHSUMED and CLEF collections. Each plot in Section 3.13 represents the AP of BM25 (x-axis) and NVSM or DRMM (y-axis).

We also employ the *Kernel Density Estimation (KDE)* [131] to estimate the *Probability Density Function (PDF)* of the AP of BM25, NVSM and DRMM models for all the topics. Then, we compute the *Kullback-Leibler Divergence (KLD)* [132] between these PDFs to get an estimate of the difference in AP distribution values for each model.

3.8 DRMM: Reproducibility of a supervised neural model

Model description. The Deep Relevance Matching Model (DRMM) [92] is a supervised system for *ad-hoc* retrieval which implements the following strategies for relevance matching:

- it considers exact matching signals between query and document terms;
- it enables a different importance weight of query terms;
- it complies with different matching requirements, i.e., *verbosity* and *scope* hypotheses. The *verbosity* hypothesis assumes that a long document is like a short document, covering a similar scope with more words. Vice versa, the *scope* hypothesis assumes that, in longer documents, only portions of the content may be relevant, the whole document is therefore not needed to be relevant for a query.

Based on word embeddings, DRMM considers local interactions between each pair of terms in a query and a document. For each query term, it maps the variable-length local interactions into a fixed-length matching histogram. Then, these matching histograms

[4] The model is available at: `https://code.google.com/archive/p/word2vec/`

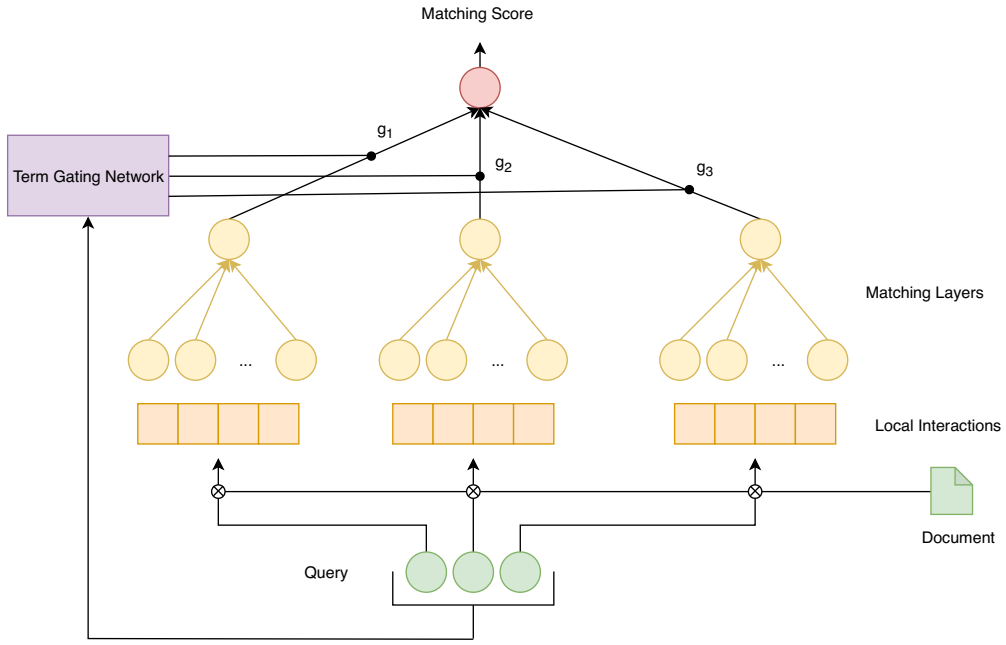


Fig. 3.1 Architecture of Deep Relevance Matching Model (DRMM).

are fed into a feed-forward neural network that outputs a matching score for each pair of query/document terms. Finally, a weighted sum of these scores for each term pair is computed in order to produce a global matching score for each query. The scheme of this architecture is depicted in Figure 3.1.

The local interactions between each query and document term are obtained computing the cosine similarity between each term vector. Scores are then aggregated in matching histograms which discretize the interval $[-1,1]$ into a set of k bins. For instance, if we take 0.5 as the bin size, each bin will contain the cosine similarity scores respectively in the intervals: $[-1,-0.5)$, $[-0.5,0.5)$, $[0.5,1)$. The interval $[1,1]$, considers the exact match scores in a separate bin. The authors propose three different ways to map the values in the matching histograms:

- Count-based Histogram (CH): considers the count of local interactions in each bin as the histogram value;
- Normalized Histogram (NH): normalizes the count value in each bin by the total count;
- LogCount-based Histogram (LCH): applies logarithm over the count value in each bin.

In our experiments and in the reference paper, the number of bins considered is 30, and we only evaluate the results of the best performing matching-histograms configuration which is LCH.

The feed-forward network, in its best-performing configuration, has two layers, the first one of 5 nodes and the second one of a single node, both using *tanh* as the activation function and a bias term.

The matching scores returned by the network are then weighted with coefficients computed by a term gating network. The term gating network optimizes the function reported below in order to estimate the best values of the coefficients g_i for the aggregation of the matching scores at the query level:

$$g_i = \frac{exp(w_g x_i^{(q)})}{\sum_{j=1}^{M} exp(w_g x_j^{(q)})},\ i = 1,...,M, \tag{3.1}$$

where w_g is the weight vector of the term network, and $x_i^{(q)},\ i = 1,...,M$ denotes the i-th query term. The authors developed different types of weighting functions which require different input values:

- Term Vector (TV): in this case, $x_i^{(q)}$ denotes the i-th query term vector, and w_g is a weight vector of the same size of the term vectors;
- Inverse Document Frequency (IDF): in this case, $x_i^{(q)}$ denotes the inverse document frequency of the i-th query term, and w_g is a coefficient with a single parameter.

From the experiments in [92], the best performing approach to compute these weights is the second one. For this reason, we employ the IDF weighting scheme in our tests.

Finally, the training of the system is performed minimizing the following hinge loss:

$$\mathscr{L}_\Theta(q, d^+, d^-) = max(0, 1 - s(q, d^+) + s(q, d^-)), \tag{3.2}$$

where d^+ is a document ranked higher than d^- given a query q, Θ represent the system parameters to be optimized and $s(\cdot)$ is the function that computes the matching scores.

Evaluation Measures. For this reproducibility experiment we consider the same evaluation measures adopted in the DRMM reference paper [92] and described in Section 2.6.1: MAP, *Normalized Discounted Cumulated Gain at Cutoff 20 (nDCG@20)* and *Precision at Cutoff 20 (P@20)*.

Model configuration. The authors of DRMM share the input data and the implementation of the system in a public repository. However, in order to replicate the original results and to generalize DRMM to other collections, we defined a new document pre-processing

pipeline and developed a new training script for the WEs required by the retrieval model. We considered only the best configuration of the model with the LCH matching histograms with 30 bins and the IDF coefficients in the term gating network. Finally, we set the first feed-forward layer size to 5. This is the default configuration of the network as described in the reference paper [92].

Moreover, since DRMM performs a re-ranking of a pre-existing run, we compute it using a modified version of Terrier v.4.1 [133] [5] with the *Query Likelihood Model* (QLM) with Dirichlet smoothing [126], the INQUERY stop list [129] and Krovetz stemmer [130] as done in the reference paper [92]. We do not perform any optimization of the parameters of the model and keep the Terrier default smoothing value $\mu = 2500$. We also keep this experimental setup for all the experiments with DRMM throughout the paper.

The most critical element among the inputs of DRMM is the set of WEs of terms in the corpus. The authors share a pre-trained Word2Vec model [6] which contains the required WEs trained on the collections used in their original experiments. However, to replicate their experiments on the Robust04 collection – and to use DRMM on other collections – we train a new Word2Vec model following their instructions with the Word2Vec implementation available in Gensim. The model was trained as described in Section 3.7, in the embedding based evaluation paragraph. The Word2Vec hyperparameters are the same as the ones indicated in the reference paper. However, differently from what stated in [92], we experimented with different number of training epochs, and employed the gradient decay option available in the Word2Vec Gensim implementation. The corpus on which we evaluate the model is the Robust04 collection, where we consider the complete set of 249 topics. Finally, we evaluate DRMM considering the *title* (T) and the *description* (D) fields as done in the experiments of the reference paper [92].

Experimental Results. In Table 3.1, we report the experimental results we obtain using the shared DRMM code with our document pre-processing pipeline. In this experiment we also used (i) the Word2Vec embeddings trained with Gensim on the Robust04 collection (Gensim WE), and (ii) the WEs model shared in the official DRMM repository (Original WE). [6] We also compare our performance to those of the QLM model implementation in Terrier and the shared results in the DRMM reference paper [92]. From the results reported in Table 3.1, we observe that our document pre-processing pipeline and the strategy we adopted for the training of the WEs leads to a similar performance to the one reported in the DRMM reference paper [92].

[5] The modified version of Terrier we used for our experiments is available at the url: `https://github.com/gridofpoints`.

[6] `https://github.com/faneshion/DRMM`

	Robust04 (T)			Robust04 (D)		
	MAP	nDCG@20	P@20	MAP	nDCG@20	P@20
QLM (original)	0.253	0.415	0.369	0.246	0.391	0.334
QLM (reproduced)	0.248 (-0.005)	0.415 (0.000)	0.355 (-0.014)	0.246 (0.000)	0.392 (+0.001)	0.326 (-0.008)
DRMM (original)	**0.279**	0.431	**0.382**	**0.275**	**0.437**	**0.371**
DRMM (Gensim WE)	0.268 (-0.011)	0.441 (+0.010)	0.376 (-0.006)	0.249 (-0.026)	0.411 (-0.026)	0.343 (0.028)
DRMM (original WE)	0.270 (-0.009)	**0.442** (+0.011)	0.377 (-0.005)	0.252 (-0.023)	0.415 (-0.022)	0.347 (-0.024)

Table 3.1 Results of our reproducibility experiments of DRMM. **Bold** values represent the highest scores among the models. We report between parentheses the difference between the reproduced results and those reported in the DRMM reference paper – QLM (original) and DRMM (original).

In general, we obtain slightly lower MAP, nDCG@20 and P@20 values than the ones reported in [92] when considering only the *title* (T) or *description* (D) fields alone, with a larger difference in the second case. Interestingly, if we repeat the experiments considering both the title and description (TD) fields of the topics, we obtain better rankings than the ones computed using only the title or description – in this case, we obtain MAP, nDCG@20 and P@20 values respectively of 0.279, 0.451 and 0.386.

Given our experimental results, we consider our document pre-processing pipeline and WE training strategy as reliable enough to reproduce the DRMM performance.

Discussion. From the results we obtained in these reproducibility experiments, we found a sizeable impact of the WEs quality on the performance of the whole system. We highlight that [92] lacks a sufficiently detailed description of the embedding training process for the replication of the results. In fact, we had to perform numerous experiments with different combinations of parameters in order to train a WE model comparable to the one in [92]. We first performed the training of the WE model with the official Word2Vec package shared by Google and the hyperparameter configuration indicated in Section 3.7. In this case, we obtained a MAP value of 0.249 on Robust04 (T). Afterwards, we tried the Word2Vec implementation available in Gensim – the one we later decided to adopt for all of our experiments – that with the same hyperparameters configuration led to a MAP of 0.268 in the same experiment.

For these reasons, we conclude that sharing the training script for the WEs model and specifying the library used are important requirements to reproduce the results of a NIR system.

Another issue we encountered in the reproducibility process concerns the preparation of the other input data required by DRMM. The shared implementation of the algorithm requires seven files: a run in TREC format to be re-ranked; a WE model to be used by the system; a file containing the document and corpus frequency for each term in the collection;

a file containing each document of the corpus with its identifier (the same used in the run to rerank), its length, and the frequency of each term in it; a file with the ideal discounted cumulative gain value for each considered topic; a file with the list of terms for each topic along with the topic identifier (the same used in the run to re-rank); the relevance judgments in TREC format for the given topics and the documents in the collection. However, the authors do not share a tool or describe, with enough detail, the process employed to compute the input in this format. Therefore, a few assumptions about the preprocessing steps to be applied to the collection were necessary. For instance, simply whitespace tokenizing, applying stemming and removing stopwords from the collection, as indicated in the reference paper, leads to the addition of great noise to the input of the system. For this reason, we first extract the text from the TREC-formatted documents, remove the tags, remove all punctuation, and finally perform the tokenization, stemming and stopwords removal. Nonetheless, we obtain about $0.9M$ distinct terms compared to the $0.7M$ terms present in the shared input file. Therefore, applying more aggressive preprocessing steps on the collection might lead to further performance improvements of DRMM.

In conclusion, we believe that for the authors of NIR research papers it would be a a good practice to share the scripts and the libraries used during document pre-processing, and, when required, also for the training of WEs models. This would allow a better and more straightforward reproducibility of their experiments. In fact, the pre-processing step is often underspecified, despite its sizeable impact on the global performance of NIR models.

3.9 NVSM: Reproducibility of an unsupervised neural model

Model description. The Neural Vector Space Model (NVSM) [2] extends two unsupervised representation learning models for expert [101] and product [100] search to *ad-hoc* retrieval. NVSM jointly learns distinct word and document representations by optimizing an unsupervised loss function which minimizes the distance between sequences of n words (i.e. n-grams) and the documents containing them. Such optimization objective imposes that n-grams extracted from a document should be predictive of that document. Unlike the models from which it derives, NVSM integrates a notion of *term specificity* [134, 135] in the learning process of word and document representations. In fact, while optimizing the n-gram representations to be close to the corresponding documents, words that are discriminative for the target documents learn to contribute more to the n-gram representations. Therefore, words associated with many documents will be neglected due to low predictive power. After training, the learned word and document representations are used to perform retrieval. Queries are seen as n-grams and matched against documents in the feature space. Documents

are then ranked in decreasing order of the cosine similarity computed between query and document representations. [7] A detailed description of NVSM components follows.

Given a document collection D and a word vocabulary V, the model considers the vector representations $\vec{w}_i \in \mathbb{R}^{k_w}$ and $\vec{d}_j \in \mathbb{R}^{k_d}$ for $w_i \in V$ and $d_j \in D$, respectively, where k_w and k_d denote the dimensionality of word and document representations. Due to the different dimensionality of word representations $\vec{w}_i$ and document representations $\vec{d}_j$, the model requires a transformation $f : \mathbb{R}^{k_w} \to \mathbb{R}^{k_d}$ from the word feature space to the document feature space. The considered transformation is linear:

$$f(\vec{x}) = W\vec{x}, \tag{3.3}$$

where $\vec{x}$ is a k_w-dimensional vector and W is a $k_d \times k_w$ parameter matrix that is learned using gradient descent. A sequence of n words (i.e. an n-gram) $(w_{j,i})_{i=1}^{n}$ extracted from a document d_j is obtained by averaging its constituent word representations as follows:

$$g\left((w_{j,i})_{i=1}^{n}\right) = \frac{1}{n}\sum_{i=1}^{n} \vec{w_{j,i}}. \tag{3.4}$$

Representations of words and documents are learned using mini-batches B of m n-gram/document pairs such that an n-gram representation is projected close to the document that contains it.

During training, an auxiliary function that L2-normalizes a vector of arbitrary dimensionality is further introduced:

$$norm(\vec{x}) = \frac{\vec{x}}{||\vec{x}||} \tag{3.5}$$

Therefore, the projection of an n-gram into the k_d-dimensional document feature space can be written as the following composition function:

$$\tilde{T}((w_{j,i})_{i=1}^{n}) = (f \circ norm \circ g)((w_{j,i})_{i=1}^{n}). \tag{3.6}$$

By estimating the per-feature sample mean and variance over batch B, the standardized projection of the n-gram representation is obtained as follows:

$$T((w_{j,i})_{i=1}^{n}) = \text{hard-tanh}\left(\frac{\tilde{T}((w_{j,i})_{i=1}^{n}) - \hat{\mathbb{E}}[\tilde{T}((w_{j,i})_{i=1}^{n})]}{\sqrt{\hat{\mathbb{V}}[\tilde{T}((w_{j,i})_{i=1}^{n})]}} + \beta\right). \tag{3.7}$$

The n-gram representation is optimized to be close to the corresponding document. The composition function g in combination with the L2-normalization *norm* causes words to

[7] Note that NVSM performs retrieval and then ranking on the whole document collection

compete for contributing to the resulting n-gram representation. Therefore, words that are discriminative for the target document learn to contribute more to the n-gram representation, and consequently, the L2-norm of the representations of discriminative words is larger than the L2-norm of non-discriminative words. This incorporates a notion of term specificity into the model. Moreover, standardization forces n-gram representations to distinguish themselves solely in the dimensions that matter for matching.

The similarity of two representations in latent vector space is defined as:

$$P(\mathscr{S}|d_j,(w_{j,i})_{i=1}^n) = \sigma(\vec{d}_j \cdot T((w_{j,i})_{i=1}^n)), \tag{3.8}$$

where $\sigma(t) = \frac{1}{1+\exp(-t)}$ denotes the sigmoid function and $\mathscr{S}$ is a binary indicator that states whether the representation of document d_j is similar to the projection of its n-gram $(w_{j,i})_{i=1}^n$ or not. The probability of a document d_j, given its n-gram $(w_{j,i})_{i=1}^n$, is then approximated by uniformly sampling z contrastive examples [136]:

$$\log \tilde{P}(d_j|(w_{j,i})_{i=1}^n)) = \frac{z+1}{2z}\left(z\log P(\mathscr{S}|d_j,(w_{j,i})_{i=1}^n)) + \sum_{\substack{k=1,\\ d_k\sim U(D)}}^{z} \log(1.0-P(\mathscr{S}|d_k,(w_{j,i})_{i=1}^n)))\right), \tag{3.9}$$

where $U(D)$ represents the uniform distribution over documents D used to obtain contrastive examples. Then, the loss function used to optimize the model, averaged over the instances in batch B, is:

$$L(\theta|B) = -\frac{1}{m}\sum_{j=1}^{m} \log \tilde{P}(d_j|(w_{j,i})_{i=1}^n) + \frac{\lambda}{2m}\Big(\sum_{i=1}^{|V|}||\vec{w}_i||_2^2 + \sum_{j=1}^{|D|}||\vec{d}_j||_2^2 + ||W||_F^2\Big), \tag{3.10}$$

where θ is the set of parameters $\{\vec{w}_i\}_{i=1}^{|V|}$, $\{\vec{d}_j\}_{j=1}^{|D|}$, W, β and λ is a weight regularization hyperparameter.

After training, a query q is projected into the document feature space by the composition of f and g: $(f \circ g)(q) = h(q)$. The matching score between a document d_j and a query q is given by the cosine similarity between their representations in document feature space:

$$score(q,d_j) = \frac{h(q)^\intercal \cdot \vec{d}_j}{||h(q)||_2\,||\vec{d}_j||_2}. \tag{3.11}$$

Documents are ranked in decreasing order of $score(q,d_j)$ for a query q.

Evaluation measures. In this reproducibility experiment, we employ the same measures reported in [2] to evaluate retrieval effectiveness: MAP, *Normalized Discounted Cumulated Gain at Cutoff 100 (nDCG@100)*, and *Precision at Cutoff 10 (P@10)*. To test statistical significance, we perform a two-tailed paired Student's t-test between Word2Vec (si) and NVSM as in [2].

Model configuration. We re-implement NVSM from scratch in Python, relying on widely-used and consolidated libraries. We employ Whoosh, [8] a fast Python search engine library, to index document collections. Whoosh allows easy access to the underlying tokenized document collections, providing the same functionalities as `pyndri` [137] which was used in the reference paper. During indexing and querying, stopwords are removed using the Indri stoplist [9] and no stemming is performed. We implement the NVSM architecture using TensorFlow.

One of the biggest challenges we found to reproduce the results presented in [2] lies in the choice of the model's parameters and hyperparameters. The reference paper and the associated GitHub repository [10] lack a comprehensive description of all the parameters and hyperparameters used by NVSM, which are crucial to reproduce results. Therefore, to select the parameters and hyperparameters used in this study, we relied on the reference paper [2], the authors' GitHub repository and an additional paper [100] on which [2] is based on. [11] For each setting, the reference source(s) are reported below.

Word vocabulary:

- Vocabulary size is limited to 2^{16} words (GitHub repository: `/scripts/functions.sh`, [100]), or to 60,000 words (NVSM reference paper [2], gihub repository: `/cpp/main.cu`);
- Words containing numbers are not considered (GitHub repository: `/cpp/main.cu`, [100]);
- Words with a document frequency lower than 2 and greater than 50% are not considered (GitHub repository: `/cpp/main.cu`).

Model parameters:

- Pseudo-random number generator seed equal to 0 (GitHub repository: `/cpp/main.cu`);

[8] `https://whoosh.readthedocs.io/en/latest/`
[9] `http://www.lemurproject.org/stopwords/stoplist.dft`
[10] `https://github.com/cvangysel/cuNVSM/`
[11] It is worth mentioning that relevant information is scattered across all these sources and none of them provide an off-the-shelf description of all the required settings

- The number of batches for a single epoch is computed as $\lceil \frac{1}{m} \sum_{d \in D} (|d| - n + 1) \rceil$, where m is the batch size (NVSM reference paper [2]);
- Word representations, document representations and the parameter matrix W are uniformly sampled in the range $[-\sqrt{\frac{6.0}{m+n}}, \sqrt{\frac{6.0}{m+n}}]$ for an $m \times n$ matrix – following the initialization scheme presented in [138] (GitHub repository, [100]).

Model hyperparameters:

- Word representation dimension $k_w = 300$ (NVSM reference paper [2]);
- Document representation dimension $k_d \in \{64, 128, 256\}$ (NVSM reference paper [2]);
- n-gram size $n \in \{4, 6, 8, 10, 12, 16, 24, 32\}$ (NVSM reference paper [2]);
- Batch size $m = 51200$ (NVSM reference paper [2]);
- The number of epochs to train the model is 15 (NVSM reference paper [2]);
- Number of negative examples $z = 10$ (NVSM reference paper [2]);
- Adam optimizer with parameters $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ (NVSM reference paper [2] reports only α, [100] reports also β_1 and β_2);
- Regularization $\lambda = 0.01$ (NVSM reference paper [2]).

For each collection considered (see Section 2.7), the given set of topics is split into validation and test sets [12] and only the *title* (T) field is used. The size of document representations $k_d \in \{64, 128, 256\}$ and of the n-grams $n \in \{4, 6, 8, 10, 12, 16, 24, 32\}$ is optimized on the validation set and then employed on the test set. The optimal hyperparameters combinations, are not reported in the reference paper.

In our experiments, we set the vocabulary size to 2^{16} and we select $k_d = 256$ according to the results reported in Fig. 3 of the reference paper [2]. Similarly, for each considered collection, we chose the n-gram size that provides the best scores in terms of MAP. The rest of the parameters and hyperparameters are kept as above. We train the model for 15 epochs and we select the model iteration that performs best in terms of MAP on the validation set. The best model is then evaluated on the test set. Tables 3.2 and 3.3 show the comparison between the results obtained with the NVSM original version and our reproduced version.

Baseline configuration. As semantic baselines, we reproduce Word2Vec (add), Word2Vec (si) and LDA. Following [2], we rely on Gensim to implement Word2Vec (skip-gram architecture) and LDA. For Word2Vec approaches, we adopted the same choices made for NVSM regarding word vocabulary, seed value, negative examples, one-sided window size (i.e., n-gram size $n/2$) and number of epochs. The embedding size is set to 256 to

[12] Splits can be found at: `https://github.com/cvangysel/cuNVSM/tree/master/resources/adhoc-splits`

be consistent with NVSM. Documents are ranked in decreasing order of cosine similarity between the document representation and the average of the WEs in the query. Once again, we selected the model iteration that performs best in terms of MAP on the validation set and we evaluated it on the test set. For LDA, we set the number of topics $K = 256$ and $\alpha = \beta = 0.1$. The model is trained until topic convergence is achieved. At query time, documents are ranked in decreasing order of the cosine similarity between the query topic distribution and the document topic distribution.

Regarding lexical baselines, we have reproduced QLM (jm) and QLM (dir). As in [2], we relied on Indri [139] to index and query the collections considered. To be consistent with semantic models, stopwords are removed using the Indri stoplist and no stemming is performed. Smoothing hyperparameters $\lambda \in \{x|k \in \mathbb{N}^+, k \leq 20, x = k/20\}$ and $\mu \in \{125, 250, 500, 750, 1000, 2000, 3000, 4000, 5000\}$ for QLM (jm) and QLM (dir), respectively, are optimized on the validation set. The comparison between the results obtained with the original baselines and our reproduced versions is also shown in Tables 3.2 and 3.3.

Rank fusion configuration. We also reproduce the fusion of three individual rankers, that is QLM (dir), Word2Vec (si) and NVSM, that provide a mixture of lexical and semantic matching. In [2], the combination of individual rankers is performed through a grid search on the weights of a linear combination using 20-fold cross validation on the topic test sets. Feature weights are swept between 0.0 and 1.0 with increments of 0.0125 on the fold training set. Individual features are normalized per query so that their values lie between 0 and 1. The coefficients configuration that achieves the highest MAP on the fold training set is selected and used to score the fold test set. During scoring of the fold test set, the pool of the top-1k documents ranked by the individual rankers is used as candidate set.

Due to the extensive memory/time requirements demanded by such approach, [13] we have to limit the number of documents in each fold of the training set to the pool of the top-1k documents produced by the individual rankers. In addition, feature weights are swept between 0.0 and 1.0 with increments of 0.1, which is the minimum step we could support on our machine. Along with the supervised approach presented in [2], we also employ three classic, fast rank fusion methods, first proposed in [1]: CombSUM, CombMNZ and CombANZ.

In Table 3.5, we evaluate the results obtained with our version of the supervised approach and the three classic methods on the Robust04 collection. The best approach in terms of

[13] The approach could not finish the first training fold, due to OOM, when considering the entire document collection and an incremental step of 0.0125. The machine used to run the experiments is a 2018 Alienware Area-51 with 36 cores and 64Gb of RAM

MAP is then employed on all the considered collections and compared to the supervised approach of the reference paper, as shown in Tables 3.6 3.7 and 3.8.

Experimental results. In Tables 3.2 and 3.3, we present a comparison between the versions of QLM, LDA, Word2Vec and NVSM described in the reference paper and our reproduced versions. For each collection, we report the results from the reference paper, the results obtained with our reproduced versions and the difference among them.

If we consider the lexical baselines, we see that the original and the reproduced versions present similar values, for both QLM (jm) and QLM (dir), on AP88-89, LA, Robust04 and WSJ – where an absolute difference greater than 0.01 is achieved by QLM (dir) only in the nDCG@100 on Robust04, and by QLM (jm) in the nDCG@100 and P@10 on LA and Robust04, respectively. A larger difference between the two versions of QLM (jm) and QLM (dir) can be observed on the FT and NY collections. The results obtained on FT show a marked difference between the original and the reproduced versions of QLM (jm) and QLM (dir) for nDCG@100 and P@10. The original version of QLM (jm) outperforms the reproduced version by an absolute difference of 0.019 in nDCG@100 and of 0.025 in P@10, whereas the original version of QLM (dir) outperforms the reproduced version by an absolute difference of 0.019 in the nDCG@100 and of 0.026 in the P@10. An opposite behavior is observed on the NY collection, where the reproduced versions of QLM (jm) and QLM (dir) outperform the original versions in all measures, with an absolute difference greater than 0.02 for the MAP and nDCG@100. Overall, QLM (dir) achieves the best results for all measures on FT, LA, NY, Robust04 and, only for the original version, for the nDCG@100 and P@10 on WSJ.

Regarding semantic baselines, the reproduced version of LDA performs similarly to the original version on all the collections considered. The most notable exceptions are the nDCG@100 and P@10 values on the NY collection, where the reproduced LDA model outperforms the original one with an absolute difference of 0.026 and 0.042, respectively. Concerning Word2Vec-based models, the reproduced version of Word2Vec (add) outperforms the original version on AP88-89, FT, WSJ, and NY according to all of the performance measures, whereas the original version achieves better results on LA and Robust04. In relation to performance difference, a more marked gap is found between the two versions only on the LA and NY collections, with differences of $+0.030$ in MAP, $+0.047$ in nDCG@100 and $+0.043$ in P@10 on LA, and of -0.036 in nDCG@100 and -0.036 in P@10 on NY. A similar trend can also be observed for Word2Vec (si), where the reproduced version outperforms the original version on AP88-89, FT, WSJ, and NY, while the original version achieves better results on LA and Robust04. The absolute differences between the two

versions are lower than or close to 0.02 on all collections, except for the NY and LA. In particular, the absolute differences are higher than 0.02 for all measures on the NY collection, with a difference of -0.064 for P@10 – which is the highest (absolute) difference amongst all measures and collections. It is also worth mentioning that, by achieving a score of 0.284 for P@10 on the NY collection, the reproduced version of Word2Vec (si) closes the gap with NVSM and results in a competitive neural baseline.

Regarding NVSM, the results obtained with the reproduced version are close to those reported in the reference paper. Considering the performance difference, the only measure that presents an absolute difference greater than 0.02 is P@10 on the WSJ collection (-0.038). On AP88-89 and NY, the absolute differences are lower than 0.01 for all the measures. NVSM outperforms the semantic baselines on all the considered collections except on the NY, where Word2Vec (si) shows better performance in MAP and nDCG@100. Furthermore, NVSM achieves the best results overall on AP88-89 and WSJ.

The results reported in Tables 3.2 and 3.3 indicate that we successfully reproduced NVSM and that the baselines adopted in the original paper are generally aligned with their reproduced versions. The optimal n-gram size and the epoch at which we obtain the best model for NVSM are indicated in Table 3.4, for each collection considered.

In Table 3.5, we present the results obtained with the reproduced version of the supervised approach and the three classic methods: CombSUM, CombMNZ and CombANZ, applied on the Robust04 collection. For each combination of QLM (dir) with Word2Vec (si) and NVSM, we can observe that the reproduced version of the supervised approach performs consistently worse than its original version. Since the two versions of QLM (dir) perform similarly, it indicates that we could not successfully reproduce the supervised rank fusion approach presented in [2]. Overall, the best method is CombSUM and the worst is CombANZ. Therefore, we employ CombSUM on all the considered collections and compare its performance to the original supervised rank fusion approach in Tables 3.6 3.7 and 3.8.

In Table Tables 3.6 3.7 and 3.8, we can observe that AP88-89 is the only collection where CombSUM achieves similar results as those of the supervised approach – with differences between the two versions of QLM(dir)+Word2Vec(si)+ NVSM of $+0.015$ for MAP, $+0.005$ for nDCG@100 and $+0.005$ for P@10. Overall, CombSUM shows positive performance gains in all the combinations of the individual rankers on AP88-89, FT and WSJ. The performance gains on LA are positive for QLM(dir)+NVSM and QLM(dir)+Word2Vec(si)+NVSM, whereas they are negative for QLM(dir)+Word2Vec(si). In particular, the CombSUM version of QLM(dir)+NVSM consistently outperforms the original supervised approach on all the measures. On Robust04, the only combination that improves over the baseline is QLM(dir)+NVSM. However, when compared to the original supervised approach, the Comb-

SUM version of QLM(dir)+NVSM presents a performance gap of about 0.020 on all the measures. Regarding the NY collection, each combination of individual rankers using CombSUM achieves lower performances than the baseline. In particular, the difference between the original supervised approach and the CombSUM version of QLM(dir)+Word2Vec(si)+NVSM is $+0.100$ for P@10.

Discussion. The results reported in Tables 3.2 and 3.3 illustrate that the reproduced and original versions of NVSM perform similarly. Therefore, based on the parameter and hyperparameter choices presented in the model configuration, we can reproduce the results of the reference paper. However, we have had to look into different sources to get appropriate values, since the configuration data are scattered across the reference paper, the GitHub repository and a previous paper [100]. Furthermore, the lack of information regarding which are the best hyperparameters to use, for the results reported in Table 2 of the reference paper, has led us to identify such hyperparameters differently. We relied on Fig. 3 from the reference paper, which allowed us to identify the optimal choices and to obtain results with the reproduced version of NVSM that are close to those reported in [2].

Another critical aspect when reproducing NVSM is the lexicon. For example, when considering the Robust04 collection, NVSM does not retrieve any document for the following four topics: topic 312 "Hydroponics"; topic 316 "Polygamy Polyandry Polygyny"; topic 348 "Agoraphobia" and topic 379 "mainstreaming". If we analyze the content of such topics, we see that three out of four topics are composed of a single word. Therefore, since the word vocabulary does not contain any of those query terms, NVSM cannot retrieve any document for them. To faithfully reproduce results, it is pivotal to know the exact lexicon used in the original paper. Otherwise, we cannot identify whether the differences between the original and the reproduced versions are related to implementation nuances or different preprocessing steps.

Regarding lexical baselines, we relied on the same search engine, i.e. Indri, and performed the same operations reported in [2] for indexing, querying and hyperparameters optimization. The differences between the original and the reproduced versions of QLM (jm) and QLM (dir) might lie on the different tokenization process applied to topics. In fact, we relied on Indri for both indexing and querying, whereas in [2] `pyndri` is used to perform querying.

Regarding semantic baselines, for LDA, we followed the same configuration presented in [2]. However, the Gensim implementation of LDA presents many more parameters than those mentioned in the reference paper. Thus, not knowing which values to assign to these parameters prevents the reproducibility of the results. For what concerns Word2Vec-based retrieval approaches, we adopt the same hyperparameter values of NVSM on every collection.

The resulting versions of Word2Vec (si) and Word2Vec (add) present sizable differences compared to the original ones. Most likely, our hyperparameter choices are different than those used in [2], especially if we consider the results obtained with the reproduced versions on the NY collection. However, in the reference paper, there is no description of the optimal choices found for Word2Vec, nor any figure that allows us to identify a subset of candidate choices. Moreover, the same considerations that are true for NVSM about the word vocabulary are also true for Word2Vec methods. Nevertheless, the results of the two-tailed paired Student's t-tests between the reproduced versions of Word2Vec (si) and NVSM are consistent with those presented in [2]. The only notable variations are in AP88-89 – where no statistical difference is found between the reproduced versions of Word2Vec (si) and NVSM – and in WSJ – where for nDCG@100 there is a statistical difference between the two reproduced versions.

Regarding rank fusion, the main issues relate to the extensive memory/time requirements of the supervised rank fusion approach. The choices we made to reproduce it were insufficient to obtain comparable results as those presented in [2]. On the other hand, classic and fast rank fusion techniques like CombSUM produced mixed results, which tended to even worsen the performance of the QLM (dir) baseline on some collections. Nevertheless, the trade-off between effectiveness and efficiency provided by CombSUM shows that far less expensive fusion methods can be employed to improve performance, achieving, on some collections, performance gains similar to those reported in [2].

		AP88-89 (T)			FT (T)			LA (T)		
		MAP	nDCG@100	P@10	MAP	nDCG@100	P@10	MAP	nDCG@100	P@10
QLM(jm)	original	0.199	0.346	0.365	0.218	0.356	0.283	0.182	0.331	0.221
	reproduced	0.199	0.346	0.364	0.209	0.337	0.258	0.178	0.319	0.214
	diff.	0.000	0.000	+ 0.001	+ 0.009	+ 0.019	+ *0.025*	+ 0.004	+ 0.012	+ 0.007
QLM(dir)	original	0.216	0.370	0.392	**0.240**	**0.381**	**0.296**	**0.198**	**0.348**	**0.239**
	reproduced	0.217	0.368	0.397	**0.230**	**0.362**	**0.270**	**0.198**	**0.341**	**0.233**
	diff.	- 0.001	+ 0.002	- 0.005	+ 0.01	+ 0.019	+ *0.026*	0.000	+ 0.007	+ 0.006
LDA	original	0.039	0.077	0.078	0.009	0.028	0.013	0.004	0.015	0.010
	reproduced	0.052	0.091	0.077	0.013	0.026	0.015	0.007	0.028	0.015
	diff.	- 0.013	- 0.014	+ 0.001	- 0.004	+ 0.002	- 0.002	- 0.003	- 0.013	- 0.005
W2V(add)	original	0.216	0.370	0.393	0.125	0.230	0.195	0.105	0.212	0.159
	reproduced	0.234	0.395	0.416	0.140	0.252	0.214	0.075	0.165	0.116
	diff.	- 0.018	- *0.025*	- *0.023*	- 0.015	- *0.022*	- 0.019	+ *0.030*	+ *0.047*	+ *0.043*
W2V(si)	original	0.230	0.383	0.418	0.141	0.250	0.204	0.131	0.242	0.179
	reproduced	0.240	0.400	0.419	0.148	0.261	0.226	0.109	0.215	0.172
	diff.	- 0.010	- 0.017	- 0.001	- 0.007	- 0.011	- *0.022*	+ *0.022*	+ *0.027*	+ 0.007
NVSM	original	**0.257****	**0.418****	**0.425**	0.172**	0.302***	0.239*	0.166**	0.300***	0.209*
	reproduced	**0.257**	**0.414**	**0.429**	0.175**	0.304***	0.220	0.180***	0.316***	0.208**
	diff.	0.000	+ 0.004	- 0.004	- 0.002	- 0.002	+ 0.019	- 0.014	- 0.016	+ 0.001

Table 3.2 Result comparison between original versions of QLM, Word2Vec, and NVSM (as in [2]) and their reproduced versions. For each experimental collection, the first row reports the scores of the original model version on MAP, nDCG@100 and P@10, the second row reports the scores of the reproduced version and the third row reports the difference between original and reproduced versions; a negative difference indicates that the reproduced baselines are stronger than those employed in the reference paper. **Bold** values represent the best model (original and reproduced), whereas *italic* values represent differences greater than 0.02. A two-tailed paired Student's t-test is computed between Word2Vec (si) and NVSM. Statistical significance is marked as * for $p < 0.1$, ** for $p < 0.05$ and *** for $p < 0.01$.

		NY (T)			Robust04 (T)			WSJ (T)		
		MAP	nDCG@100	P@10	MAP	nDCG@100	P@10	MAP	nDCG@100	P@10
QLM(jm)	original	0.158	0.270	0.376	0.201	0.359	0.369	0.175	0.315	0.345
	reproduced	0.180	0.292	0.382	0.199	0.351	0.358	0.178	0.319	0.347
	diff.	- *0.022*	- *0.22*	- 0.006	+ 0.002	+ 0.008	+ 0.011	- 0.003	- 0.004	- 0.002
QLM(dir)	original	**0.188**	**0.318**	**0.486**	**0.224**	**0.388**	**0.415**	0.204	**0.351**	**0.398**
	reproduced	**0.213**	**0.343**	**0.500**	**0.222**	**0.376**	**0.411**	0.205	0.355	0.391
	diff.	- *0.025*	- *0.025*	- 0.014	+ 0.002	+ 0.012	+ 0.004	- 0.001	- 0.004	+ 0.007
LDA	original	0.009	0.027	0.022	0.003	0.010	0.009	0.038	0.082	0.076
	reproduced	0.024	0.053	0.064	0.004	0.015	0.014	0.041	0.074	0.060
	diff.	- 0.015	- *0.026*	- *0.042*	- 0.001	- 0.005	- 0.005	- 0.003	+ 0.008	+ 0.016
W2V(add)	original	0.081	0.160	0.216	0.075	0.177	0.194	0.175	0.322	0.372
	reproduced	0.100	0.196	0.252	0.065	0.158	0.184	0.181	0.326	0.393
	diff.	- 0.019	- *0.036*	- *0.036*	+ 0.010	+ 0.019	+ 0.010	- 0.006	- 0.004	- *0.021*
W2V(si)	original	0.092	0.173	0.220	0.093	0.208	0.234	0.185	0.330	0.391
	reproduced	0.113	0.209	0.284	0.083	0.192	0.214	0.190	0.336	0.393
	diff.	- *0.021*	- *0.036*	- *0.064*	+ 0.010	+ 0.016	+ *0.020*	- 0.005	- 0.006	- 0.002
NVSM	original	0.117	0.208	0.296*	0.150***	0.287***	0.298***	**0.208****	**0.351**	0.370
	reproduced	0.110	0.205	0.290	0.138***	0.270***	0.289***	**0.213*****	**0.359*****	**0.408**
	diff.	+ 0.007	+ 0.003	+ 0.006	+ 0.012	+ 0.017	+ 0.009	- 0.005	- 0.008	- *0.038*

Table 3.3 Result comparison between original versions of QLM, Word2Vec, and NVSM (as in [2]) and their reproduced versions. For each experimental collection, the first row reports the scores of the original model version on MAP, nDCG@100 and P@10, the second row reports the scores of the reproduced version and the third row reports the difference between original and reproduced versions; a negative difference indicates that the reproduced baselines are stronger than those employed in the reference paper. **Bold** values represent the best model (original and reproduced), whereas *italic* values represent differences greater than 0.02. A two-tailed paired Student's t-test is computed between Word2Vec (si) and NVSM. Statistical significance is marked as * for $p < 0.1$, ** for $p < 0.05$ and *** for $p < 0.01$.

	n-gram size	Best epoch
AP88-89	16	13
FT	16	11
LA	10	10
WSJ	16	14
Robust04	16	11
NY	16	1

Table 3.4 NVSM optimal n-gram size and best epoch for each collection.

		Robust04 (T)		
		MAP	nDCG@100	P@10
QLM(dir)	original	0.224	0.388	0.415
	reproduced	0.222	0.376	0.411
QLM(dir)+W2V(si)	original	0.232	0.399	0.428
	reproduced	0.190	0.344	0.346
	CombSUM	**0.199**	**0.358**	**0.378**
	CombMNZ	0.191	0.353	0.366
	CombANZ	0.179	0.325	0328
QLM(dir)+NVSM	original	0.247	0.411	0.448
	reproduced	0.204	0.357	0.375
	CombSUM	**0.230**	0.391	**0.424**
	CombMNZ	0.229	**0.392**	0.419
	CombANZ	0.199	0.352	0.373
QLM(dir)+W2V(si)+NVSM	original	0.247	0.412	0.446
	reproduced	0.1836	0.336	0.344
	CombSUM	**0.206**	**0.368**	**0.385**
	CombMNZ	0.202	0.363	0.373
	CombANZ	0.175	0.323	0.338

Table 3.5 Result comparison on the Robust04 collection between our version of the supervised rank fusion approach, presented in [2], and the three classic methods originally proposed in [1]: CombSUM, CombMNZ and CombANZ. The first two rows report, for reference, the scores of the original and the reproduced versions of QLM (dir) for MAP, nDCG@100 and P@10. Then, for each combination of QLM (dir) with Word2Vec (si) and NVSM, the first two rows report the scores of the original and the reproduced versions of the supervised approach. Subsequent rows report the scores of CombSUM, CombMNZ and CombANZ, respectively. **Bold** values represent the best method amongst the supervised approach (reproduced) and the three classic methods.

		AP88-89 (T)			FT (T)		
		MAP	nDCG@100	P@10	MAP	nDCG@100	P@10
QLM(dir)	original	0.216	0.370	0.392	0.240	0.381	0.296
	reproduced	0.217	0.368	0.397	0.230	0.362	0.270
QLM(dir)+W2V(si)	original	0.279 (+29%)	0.437 (+18%)	0.450 (+14%)	0.251 (+4%)	0.393 (+3%)	0.313 (+6%)
	CombSUM	0.275 (+27%)	0.441 (+20%)	0.446 (+12%)	0.242 (+5%)	0.381 (+5%)	0.293 (+9%)
	diff.	+ 0.004	- 0.004	+ 0.004	+ 0.009	+ 0.012	*+ 0.020*
QLM(dir)+NVSM	original	0.289 (+33%)	0.444 (+20%)	0.473 (+20%)	0.251 (+4%)	0.401 (+5%)	0.322 (+9%)
	CombSUM	0.269 (+24%)	0.428 (+16%)	0.456 (+15%)	0.233 (+1%)	0.378 (+4%)	0.286 (+6%)
	diff.	*+ 0.020*	+ 0.016	+ 0.017	+ 0.018	*+ 0.023*	*+ 0.036*
QLM(dir)+W2V(si)+ NVSM	original	**0.307** (+42%)	**0.466** (+26%)	**0.498** (+27%)	**0.258** (+7%)	**0.406** (+6%)	**0.322** (+9%)
	CombSUM	**0.292** (+35%)	**0.461** (+25%)	**0.493** (+24%)	**0.244** (+6%)	**0.386** (+7%)	**0.297** (+10%)
	diff.	+ 0.015	+ 0.005	+ 0.005	+ 0.014	*+ 0.020*	*+ 0.025*

Table 3.6 Result comparison between CombSUM rank fusion [1] and the supervised rank fusion proposed in [2]. For each experimental collection, the scores of the original and the reproduced versions of QLM (dir) on MAP, nDCG@100 and P@10 are reported for reference. For each of the three combinations of QLM (dir) with Word2Vec (si) and NVSM, the first row reports the original scores of the supervised rank fusion while the second row reports the scores of the CombSUM rank fusion. The third row illustrates the difference between the original scores of the supervised approach and the scores obtained with CombSUM; a negative difference indicates that CombSUM achieves higher scores than those of the supervised approach used in the reference paper. **Bold** values represent the best method (original and reproduced), whereas *italic* values represent differences greater than 0.02. The percentage gain (or loss) over the baseline method is reported next to each rank fusion approach.

		LA (T)			NY (T)		
		MAP	nDCG@100	P@10	MAP	nDCG@100	P@10
QLM(dir)	original	0.198	0.348	0.239	0.188	0.318	0.486
	reproduced	0.198	0.341	0.233	**0.213**	**0.343**	**0.500**
QLM(dir)+W2V(si)	original	0.212 (+7%)	0.360 (+3%)	0.236 (-1%)	0.206 (+9%)	0.333 (+4%)	0.494 (+1%)
	CombSUM	0.191 (-4%)	0.326 (-4%)	0.229 (-2%)	0.194 (-9%)	0.325 (-5%)	0.436 (-13%)
	diff.	+ *0.021*	+ *0.034*	+ 0.007	+ 0.012	+ 0.008	+ *0.058*
QLM(dir)+NVSM	original	0.220 (+11%)	0.376 (+7%)	0.244 (+1%)	**0.222** (+18%)	**0.355** (+11%)	0.520 (+6%)
	CombSUM	**0.232** (+17%)	**0.381** (+12%)	**0.255** (+9%)	0.198 (-7%)	0.333 (-3%)	0.476 (-5%)
	diff.	- 0.012	- 0.005	- 0.011	+ *0.024*	+ *0.022*	+ *0.044*
QLM (dir)+W2V(si)+ NVSM	original	**0.226** (+14%)	**0.378** (+8%)	**0.250** (+4%)	**0.222** (+18%)	0.353 (+10%)	**0.526** (+8%)
	CombSUM	0.214 (+8%)	0.366 (+7%)	0.251 (+7%)	0.182 (-14%)	0.320 (-7%)	0.426 (-15%)
	diff.	+ 0.012	+ 0.012	- 0.001	+ *0.040*	+ *0.033*	+ *0.100*

Table 3.7 Result comparison between CombSUM rank fusion [1] and the supervised rank fusion proposed in [2]. For each experimental collection, the scores of the original and the reproduced versions of QLM (dir) on MAP, nDCG@100 and P@10 are reported for reference. For each of the three combinations of QLM (dir) with Word2Vec (si) and NVSM, the first row reports the original scores of the supervised rank fusion while the second row reports the scores of the CombSUM rank fusion. The third row illustrates the difference between the original scores of the supervised approach and the scores obtained with CombSUM; a negative difference indicates that CombSUM achieves higher scores than those of the supervised approach used in the reference paper. **Bold** values represent the best method (original and reproduced), whereas *italic* values represent differences greater than 0.02. The percentage gain (or loss) over the baseline method is reported next to each rank fusion approach.

		Robust04 (T)			WSJ (T)		
		MAP	nDCG@100	P@10	MAP	nDCG@100	P@10
QLM(dir)	original	0.224	0.388	0.415	0.204	0.351	0.398
	reproduced	0.222	0.376	0.411	0.205	0.355	0.391
QLM(dir)+W2V(si)	original	0.232 (+3%)	0.399 (+2%)	0.428 (+2%)	0.254 (+24%)	0.410 (+16%)	0.454 (+13%)
	CombSUM	0.199 (-10%)	0.358 (-5%)	0.378 (-8%)	0.238 (+16%)	0.399 (+12%)	0.449 (+15%)
	diff.	*+ 0.033*	*+ 0.041*	*+ 0.050*	+ 0.016	+ 0.011	+ 0.005
QLM(dir)+NVSM	original	**0.247** (+10%)	0.411 (+6%)	**0.448** (+7%)	0.248 (+21%)	0.396 (+12%)	0.425 (+6%)
	CombSUM	**0.230** (+4%)	**0.391** (+4%)	**0.424** (+3%)	0.244 (+19%)	0.403 (+14%)	0.443 (+13%)
	diff.	+ 0.017	*+ 0.020*	*+ 0.024*	+ 0.004	- 0.007	- 0.018
QLM(dir)+W2V(si)+ NVSM	original	**0.247** (+10%)	**0.412** (+6%)	0.446 (+7%)	**0.271** (+32%)	**0.426** (+21%)	**0.456** (+14%)
	CombSUM	0.206 (-7%)	0.369 (-2%)	0.385 (-6%)	**0.251** (+22%)	**0.416** (+17%)	**0.455** (+16%)
	diff.	*+ 0.041*	*+ 0.043*	*+ 0.061*	*+ 0.020*	+ 0.010	+ 0.001

Table 3.8 Result comparison between CombSUM rank fusion [1] and the supervised rank fusion proposed in [2]. For each experimental collection, the scores of the original and the reproduced versions of QLM (dir) on MAP, nDCG@100 and P@10 are reported for reference. For each of the three combinations of QLM (dir) with Word2Vec (si) and NVSM, the first row reports the original scores of the supervised rank fusion while the second row reports the scores of the CombSUM rank fusion. The third row illustrates the difference between the original scores of the supervised approach and the scores obtained with CombSUM; a negative difference indicates that CombSUM achieves higher scores than those of the supervised approach used in the reference paper. **Bold** values represent the best method (original and reproduced), whereas *italic* values represent differences greater than 0.02. The percentage gain (or loss) over the baseline method is reported next to each rank fusion approach.

3.10 Comparison with other lexical and semantic retrieval models

In this section, we compare the performance of NVSM and DRMM (re-ranking over QLM (dir)) to other lexical and semantic models considering the Robust04 (T) and NY (T) collections. Other retrieval models we consider in this comparison are: QLM (dir), DFR, BM25, TF-IDF (all with Krovetz stemmer) and Word2Vec (add and self-information versions). We also employ NVSM to re-rank the same QLM (dir) runs used by DRMM in its experiments where the objective is to investigate how well a neural unsupervised model like NVSM performs within a re-ranking scenario. Finally, we perform a statistical significance analysis with Tukey's T test to assess the statistical significance of performance differences of the retrieval models. The evaluation measures used in this set of experiments are: MAP, nDCG@100 and P@10.

Experimental results. The results in Table 3.9 show that the lexical models considered perform better than all the Word2Vec-based approaches or the NIR models on the NY collection. On the other hand, we also notice that DRMM outperforms both the lexical models and NVSM on the Robust04 collection. Instead, NVSM always performs better than the semantic matching models, but is never competitive with other lexical models nor with DRMM. In the re-ranking task, NVSM outperforms DRMM on the NY collection, but not on the Robust04. Furthermore, we observe that performing re-ranking with NVSM on QLM (dir) always leads to a performance improvement over NVSM alone. However, the re-ranking produced by NVSM significantly worsens the performance of the QLM (dir) baseline on both collections.

For reference, we report the best values for MAP and P@10 obtained in the TREC 2004 Robust Retrieval Track [140] on the Robust04 collection, and the best value for MAP in the TREC 2017 Common Core Track [141, 142] on the NY collection. On Robust04 (T), the best value for MAP is 0.333 and for P@10 is 0.513. On NY, the best value for MAP is 0.538.

Significance tests. In Figure 3.2, we report the results of Tukey's T test on the runs produced by the systems in Table 3.9. Firstly, we notice that on the NY collection there is no statistical difference between considered lexical models (i.e. QLM (dir), DFR, TF-IDF and BM25). Additionally, pure lexical models always perform statistically better than semantic models (i.e. Word2Vec (add), Word2Vec (si), QLM(dir)/NVSM, DRMM and NVSM). If we consider the tests on the Robust04 collection, we observe that DRMM belongs to the top group and statistically outperforms the other semantic matching models, including the

	NY (T)			Robust04 (T)		
	MAP	nDCG	P@10	MAP	nDCG	P@10
QLM(dir)	0.232	0.370	**0.522**	0.248	0.411	0.424
BM25	**0.234**	0.347	0.468	0.242	0.404	0.431
TF-IDF	0.228	**0.499**	0.472	0.242	0.405	0.431
DFR	0.225	0.350	0.478	0.227	0.390	0.425
Word2Vec (add)	0.100	0.196	0.252	0.062	0.150	0.175
Word2Vec (si)	0.113	0.209	0.284	0.081	0.185	0.205
DRMM	0.141	0.211	0.274	**0.268**	**0.435**	**0.455**
NVSM	0.110	0.205	0.290	0.139	0.269	0.279
QLM(dir)/NVSM	0.152	0.252	0.328	0.157	0.289	0.287

Table 3.9 Results comparison between reproduced versions of DRMM, NVSM and QLM(dir)/NVSM and other lexical and neural baselines: DRM, BM25, TF-IDF, QLM and Word2Vec. For each experimental collection used, the scores on MAP, nDCG at rank 100 AND P@10 are reported for each model. **Bold** values represent the highest scores among the models.

re-ranking of the QLM (dir) run performed by NVSM – i.e., when NVSM performs the same task as DRMM. However, it is worth mentioning that QLM(dir)/NVSM is an unsupervised re-ranking model, whereas DRMM is a supervised one. Therefore, while NVSM performs re-ranking relying only on word and document representations learned from the collection, DRMM exploits relevance judgments to learn how to rank documents given a query. These observations apply to all the performance measures considered – i.e. MAP, nDCG@100, and P@10.

Discussion. From the results reported in Table 3.9, our first observation has to do with the performance of DRMM which differs in the two considered collections. In fact, DRMM improves the ranking of QLM (dir) and it is the best system on the Robust04 collection; but, it worsens the ranking computed by the QLM (dir) baseline, provided in input on the NY collection. This result reflects one of the weaknesses of supervised NIR models: i.e., their inability to generalize when trained on a limited number of topics. Compared to Robust04, which presents 249 topics, the NY collection has only 50 topics – about one fifth of the topics of Robust04. Therefore, on the NY collection DRMM suffers from the lack of topics to learn from, compared to the large size of the collection. In fact, the large number of documents – combined with the intrinsic characteristics of the collection and the queries – generates a wide variety of matching signals to be interpreted by the model. This makes it harder for DRMM to learn how to discriminate between relevant and non-relevant documents without seeing larger parts of the collection during training. This claim is also confirmed by

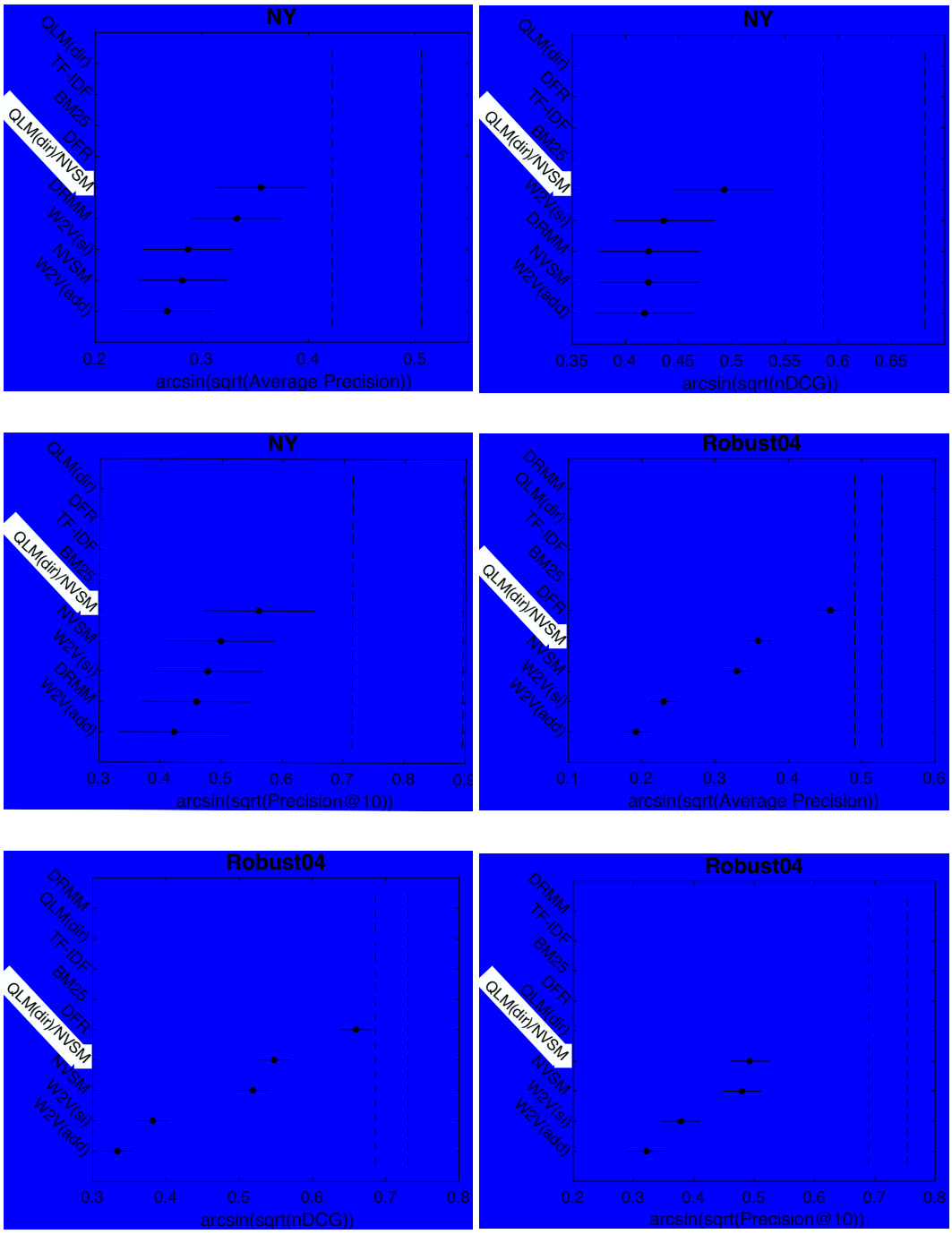


Fig. 3.2 Significance tests for the results reported in Table 3.9 (top group highlighted). All pairwise comparisons are calculated with Tukey's HSD confidence intervals and a significance level $\alpha = 0.05$. Each row depicts the comparisons made for MAP, NDCG at rank 100 and P@10 for a specific collection.

our experiments on the WT2g collection in Section 3.11. Indeed, the WT2g collection has the same number of topics of the NY collection, but contains seven times less documents. This difference in size reduces the effect of the lack of training topics on DRMM which performs similarly to other lexical models on this collection. Conversely, on the Robust04 collection, where the model is trained on a larger number of topics, DRMM outperforms all the other baselines. On the other hand, the re-ranking performed by NVSM always worsens the performance of the QLM (dir) baseline. This suggests that NVSM does not effectively re-rank the results provided by a lexical model like QLM (dir). As shown in Tables 3.6, 3.7, 3.8, a rank fusion approach that combines lexical and semantic signals together is better suited for NVSM than a simple re-ranking approach.

When considering other state-of-the-art approaches that use different ranking strategies – such as [127], an application of BERT to IR that achieves a MAP of 0.328 on the Robust04, or the BM25+RM3 model in Anserini [117] that achieves a MAP of 0.2903 [14] – we have observed that DRMM and NVSM are not competitive with such approaches. However, these two NIR models remain relevant as they are unsupervised (NVSM) and supervised (DRMM) innovative neural architectures for document retrieval which do not rely on any auxiliary data source or component, like relevance feedback or query expansion. Indeed, the advancement proposed by DRMM and NVSM is more methodological rather than performance oriented. Nevertheless, the combination of DRMM and NVSM with pre-trained NLP models, such as BERT, or other IR techniques, such as RM3, may still lead to better performance than state-of-the-art.

3.11 Collection-based evaluation

In this set of experiments, we evaluate the ability of DRMM and NVSM models to generalize in the presence of different domains and tasks. The considered collections and topic fields include: CLEF-DE (TD), CLEF-FA (TD), CLEF-IT (TD), WT2G (TD), and OHSUMED (D). We select the above combinations of topic fields, since they are the ones that lead to the best retrieval performance on each collection and the most widely used. Additionally, we compare the performance of DRMM and NVSM to those of lexical IR models, that is BM25, DFR, TF-IDF and QLM (dir). To perform retrieval on CLEF multilingual collections, all the considered models rely on publicly available stop lists specifically defined for the target language. [15] The evaluation measures we consider for this set of experiments are:

[14] `https://github.com/castorini/anserini/blob/master/docs/experiments-robust04.md`

[15] `http://members.unine.ch/jacques.savoy/clef`

MAP, nDCG@100, and P@10. Furthermore, we rely on post-hoc Tukey's T test to assess statistical significance.

Hyperparameter tuning. The performance measures of DRMM are obtained according to the steps described in Section 3.7, where a re-ranking of the documents in each QLM (dir) run is performed. The main hyperparameters of DRMM are the number of bins in the matching histograms and the size of the hidden layer. The number of bins in the matching histograms should be high enough to allow the representation of different degrees of word similarity. At the same time, it should be small enough to maintain the generalization power of the model and keep computational complexity low.

We performed hyperparameter optimization on the OHSUMED and WT2g collections – which are the corpora that differ the most from the ones used in the DRMM reference paper – to assess whether the changes to the mentioned hyperparameters could improve the performance of the model. We considered all the combinations of the following hyperparameters: number of bins in the matching histograms, in the range {5, 10, 15, 20, 25, 30, 35} and hidden layer size, in the range {5, 10, 15, 20, 25}. In our experiments, we noticed a performance improvement – an increase of 0.02 or higher in MAP, nDCG@100 and P@10 – only on the WT2g collection with 20 bins in the matching histograms, and 15 units in the hidden layer. We will keep this configuration of DRMM also for the other experiments on this collection. On the other hand, we did not notice any sizable performance increase on the OHSUMED collection changing the hyperparameters of the DRMM model.

Regarding NVSM, we adopted the same experimental setup described in Section 3.9. For every collection, we optimize the following hyperparameters:

- Vocabulary size $|V| \in \{2^{16}, 2^{17}\}$;
- Document representation dimension $k_d \in \{128, 256\}$;
- n-gram size $n \in \{4, 6, 8, 10, 12, 16, 24, 32\}$;
- Batch size $m \in \{12800, 25600, 51200\}$;
- Regularization $\lambda \in \{0.01, 0.1, 1.0\}$.

The rest of the hyperparameters are kept as in Section 3.9. Due to the prohibitive time required to perform grid search over the hyperparameters, we first optimize the n-gram size by keeping the default values for $|V| = 2^{16}$, $k_d = 256$, $m = 51200$, and $\lambda = 0.01$. Then, for each collection we keep the n-gram size that performs best in terms of MAP and we optimize the rest of the hyperparameters. From this optimization we did not detect any improvement related to different combinations of k_d, m, and λ. However, the impact of the vocabulary size has shown to be significant on two collections: WT2g and CLEF-DE. For WT2g, NVSM goes from MAP: 0.206, nDCG@100: 0.356, and P@10: 0.370, with $|V| = 2^{16}$, to MAP:

0.225, nDCG@100: 0.380, and P@10: 0.402, with $|V| = 2^{17}$. Similarly, for CLEF-DE the model goes from MAP: 0.194, nDCG@100: 0.322, and P@10: 0.281, with $|V| = 2^{16}$, to MAP: 0.211, nDCG@100: 0.343, and P@10: 0.301, with $|V| = 2^{17}$. Therefore, we adopt $|V| = 2^{17}$ for WT2g and CLEF-DE collections and we keep the rest of the hyperparameters as default. The optimal n-gram size, vocabulary size, and the epoch at which we obtain the best model for NVSM are reported in Table 3.10 for every collection.

Experimental results. Table 3.11, illustrates the retrieval results related to the WT2g and OHSUMED collections. In such case, we first observe that DRMM always improves the ranking of the runs produced with QLM (dir). We also employed DRMM to re-rank the runs obtained with BM25 and TF-IDF, but we found no substantial improvement over the baselines. Conversely, NVSM performances are the worst overall on WT2g and outperform only those of QLM (dir) on OHSUMED. In particular, the performance gap between NVSM and all the other models is more accentuated on WT2g than on OHSUMED. This may be due to the fact that NVSM derives from two approaches that are specifically tailored to product and expert search [100, 101]. Therefore, the model has a robust domain-specific nature and for heterogeneous collections like WT2g, where documents have different contents and scopes, it generalizes worse than for homogeneous collections like OHSUMED. Indeed, NVSM achieves better results than QLM (dir) on OHSUMED for all the evaluation measures considered.

For reference, we report the best value for MAP obtained in the TREC-8 Web Track [76] on the WT2g collection, and the best value for MAP obtained on the OHSUMED collection according to [64]. These are 0.383 and 0.450 on WT2g and OHSUMED, respectively.

In Table 3.12, we report the experimental results on CLEF collections. If we consider the performances of DRMM and compare them to QLM (dir), we do not observe the same trend for all the experimental collections. DRMM outperforms QLM (dir) on CLEF-IT and CLEF-FA collections, whereas it worsens the performance of QLM (dir) on CLEF-DE. Overall, the performance of DRMM and the lexical baselines follow the same trend of the previous set of experiments, where BM25 and TF-IDF are the two best baselines. The results of NVSM also represent a similar behavior as those reported for the two previous collections when compared to BM25, TF-IDF, and DFR. Nevertheless, NVSM shows competitive results with QLM (dir) on CLEF-IT and CLEF-DE, and it outperforms QLM (dir) on CLEF-FA. When compared to DRMM, NVSM achieves lower results on CLEF-IT and CLEF-FA, but it outperforms DRMM on CLEF-DE.

For reference, we report the best values for MAP obtained in the CLEF 2006 Ad Hoc Track [71] on the CLEF-IT and CLEF-DE collections, and the best value for MAP obtained

in the CLEF 2009 Ad Hoc Track [143] on the CLEF-FA collection. On CLEF-IT (TD) and CLEF-DE (TD), the best values for MAP are 0.419 and 0.483, respectively. On CLEF-FA (TD), the best value for MAP is 0.494.

	n-gram size	Vocabulary	Best epoch
WT2g	16	131072	11
OHSUMED	16	65536	12
CLEF-IT	20	65536	14
CLEF-DE	8	131072	13
CLEF-FA	16	65536	15

Table 3.10 NVSM optimal *n*-gram size, vocabulary size, and best epoch for each collection.

	WT2g (TD)			OHSUMED (D)		
	MAP	nDCG	P@10	MAP	nDCG	P@10
QLM(dir)	0.264	0.418	0.418	0.212	0.326	0.305
BM25	**0.296**	0.454	**0.484**	0.250	0.369	0.364
TF-IDF	0.234	0.389	0.419	0.250	0.370	0.364
DFR	0.267	0.426	0.452	0.236	0.354	0.344
DRMM	0.289	**0.455**	0.434	**0.272**	**0.407**	**0.375**
NVSM	0.225	0.380	0.402	0.214	0.335	0.319

Table 3.11 Generalization experiments results on the Web (WT2g) and medical (OHSUMED) domain. The highest score amongst the models is in **bold**. The considered evaluation measures are MAP, nDCG@100 and P@10.

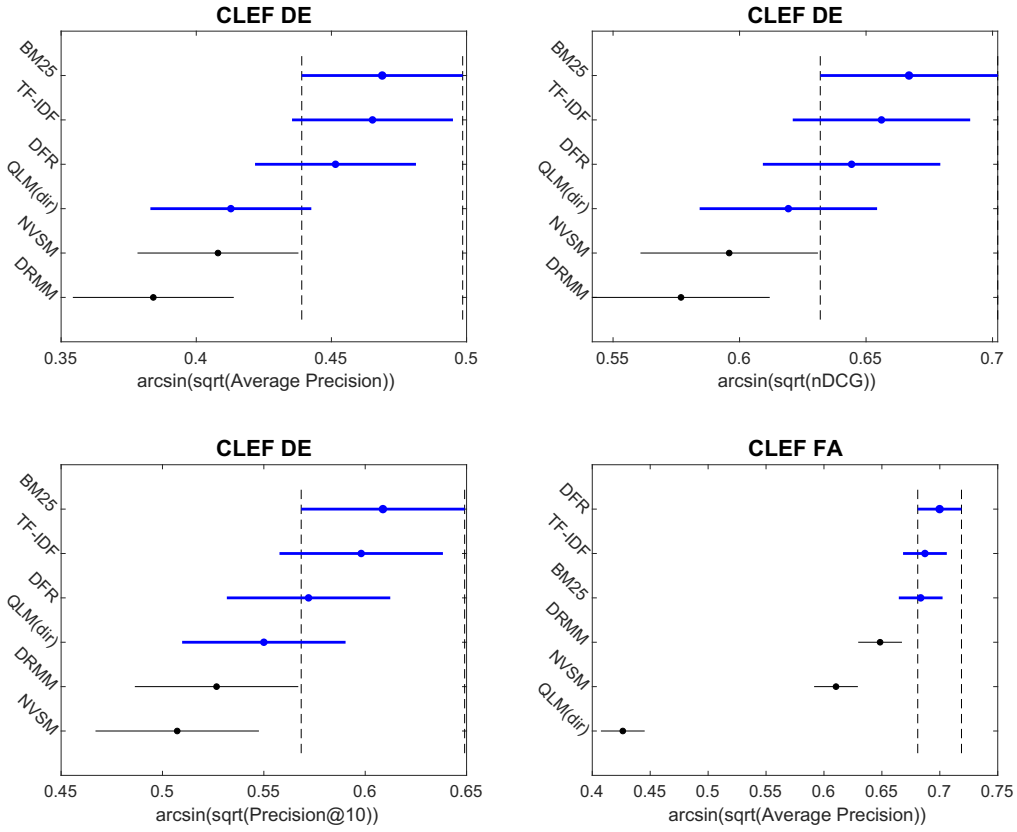
	CLEF-IT (TD)			CLEF-DE (TD)			CLEF-FA (TD)		
	MAP	nDCG	P@10	MAP	nDCG	P@10	MAP	nDCG	P@10
QLM(dir)	0.334	0.510	0.319	0.209	0.350	0.324	0.200	0.340	0.405
BM25	0.437	0.627	0.402	**0.253**	**0.395**	**0.373**	0.408	0.550	0.594
TF-IDF	**0.438**	**0.628**	**0.408**	0.251	0.392	0.367	0.411	0.553	0.601
DFR	0.415	0.605	0.393	0.241	0.382	0.350	**0.421**	**0.565**	0.619
DRMM	0.357	0.557	0.349	0.188	0.329	0.316	0.375	0.551	**0.623**
NVSM	0.345	0.522	0.317	0.211	0.343	0.301	0.342	0.494	0.567

Table 3.12 Generalization experiments results on CLEF collections in Italian (IT), German (DE) and Persian (FA). The highest score amongst the models is in **bold**. The considered evaluation measures are MAP, nDCG@100 and P@10.

Significance Tests. From Tukey's T tests conducted with a significance level $\alpha = 0.05$ reported in Figure 3.3, we observe a similar behavior as in the results reported in Tables

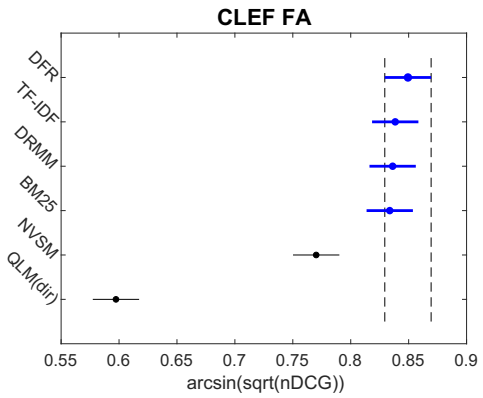
3.11 and 3.12. NVSM is never in the top group – except when we consider the P@10 on the CLEF-FA collection – while DRMM is in the top group for most of the collections (i.e. CLEF-FA, WT2g, and OHSUMED). Regarding lexical models, BM25, TF-IDF and DFR are almost always the best performing systems, while QLM (dir) is in general the worst performing one – often being out of the top group (e.g. in the CLEF-FA, CLEF-IT, and OHSUMED collections).

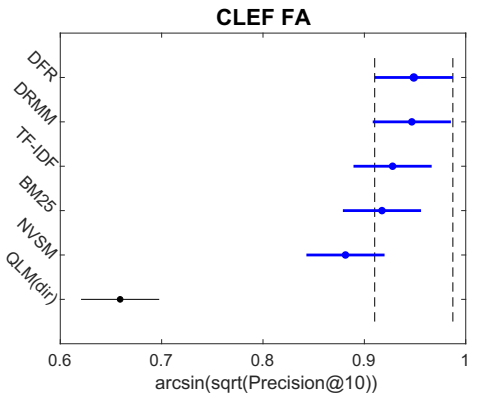
Discussion. From the experimental results described above, we can conclude that DRMM does not always statistically improve the ranking produced by the QLM (dir) baseline. The only collections where there is a statistically significant improvement of DRMM over QLM (dir) are CLEF-FA and OHSUMED. The same behavior can be found when DRMM is trained and used to re-rank runs produced by different lexical models (i.e., TF-IDF and BM25), where it often fails to produce a substantial improvement. For these reasons, it is unclear whether the improvement brought by DRMM in the re-ranking task is more attributable to DRMM itself, or to the lexical model used to compute the input run – which might be good at finding relevant documents but bad at ranking them. In fact, we observe that DRMM tends

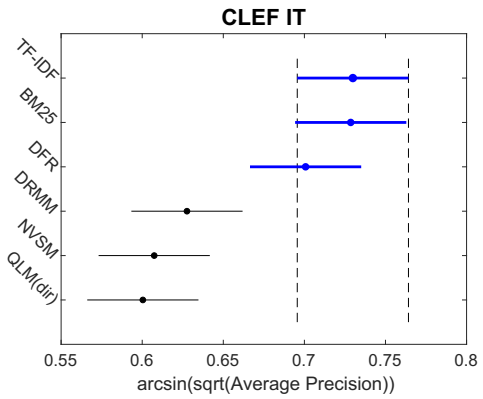


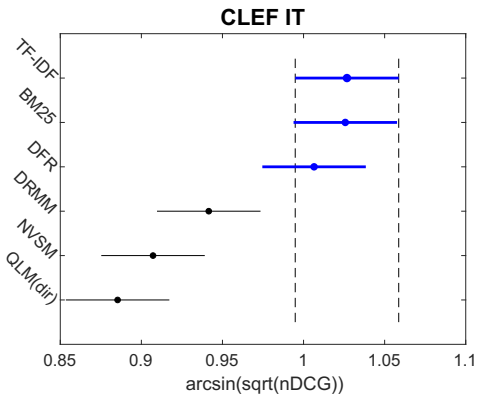
to achieve competitive performances only when those of QLM (dir) are particularly low compared to other lexical models, e.g. on the OHSUMED and CLEF-FA collections. On the other hand, if we consider WT2g or CLEF-DE, where QLM (dir) achieves performances closer to those of other lexical models, the re-ranking of DRMM is less effective or even detrimental. Indeed, DRMM worsens the QLM (dir) baseline in all measures on CLEF-DE. However, the overall results on CLEF-DE – compared to those obtained on the other CLEF collections – suggest that CLEF-DE is a more difficult collection to perform retrieval on. Nevertheless, we cannot exclude that, if more topics were provided in the training step, DRMM would improve the ranking of QLM (dir) also on this collection.

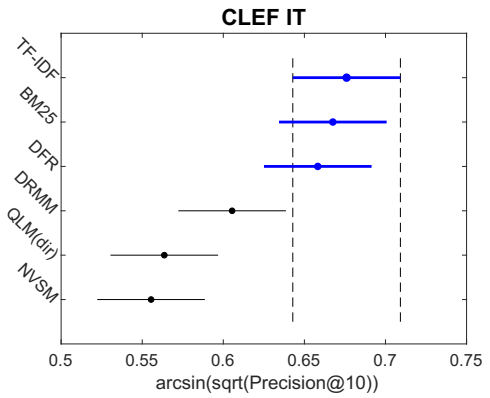
Regarding NVSM, we observe that it is not competitive with traditional lexical models. The only exception is QLM (dir), that achieves results that are comparable to or lower than NVSM on the OHSUMED and CLEF collections. Results from Tables 3.11 and 3.12 show that NVSM struggles to generalize when considering heterogeneous data. The reason why NVSM suffers more with the WT2g collection than on OHSUMED and CLEF collections may be related to its inherent domain-specific nature. Indeed, NVSM derives from [100, 101], which are two approaches targeting product and expert search, respectively.

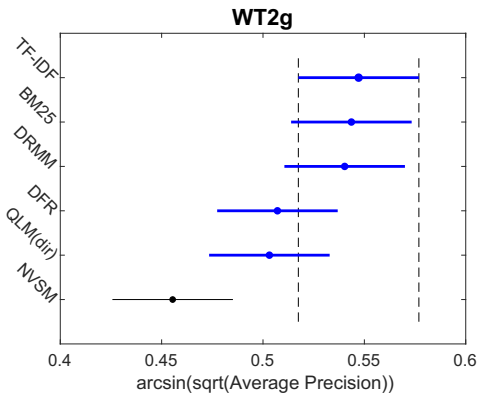

CLEF FA
DFR
TF-IDF
DRMM
BM25
NVSM
QLM(dir)
0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9
arcsin(sqrt(nDCG))

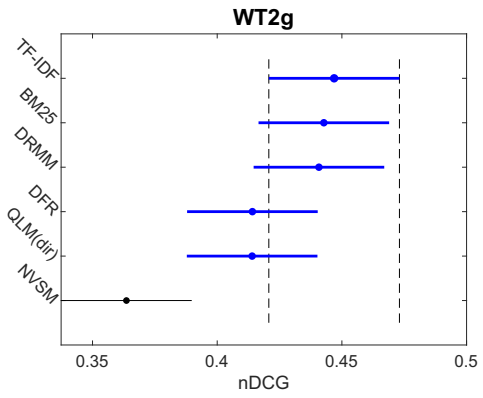

CLEF FA
DFR
DRMM
TF-IDF
BM25
NVSM
QLM(dir)
0.6 0.7 0.8 0.9 1
arcsin(sqrt(Precision@10))

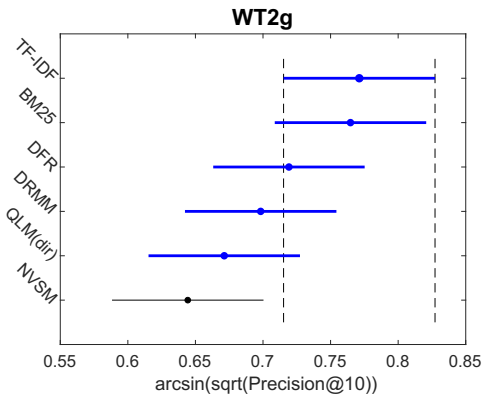

CLEF IT
TF-IDF
BM25
DFR
DRMM
NVSM
QLM(dir)
0.55 0.6 0.65 0.7 0.75 0.8
arcsin(sqrt(Average Precision))


CLEF IT
TF-IDF
BM25
DFR
DRMM
NVSM
QLM(dir)
0.85 0.9 0.95 1 1.05 1.1
arcsin(sqrt(nDCG))


CLEF IT
TF-IDF
BM25
DFR
DRMM
QLM(dir)
NVSM
0.5 0.55 0.6 0.65 0.7 0.75
arcsin(sqrt(Precision@10))


WT2g
TF-IDF
BM25
DRMM
DFR
QLM(dir)
NVSM
0.4 0.45 0.5 0.55 0.6
arcsin(sqrt(Average Precision))


WT2g
TF-IDF
BM25
DRMM
DFR
QLM(dir)
NVSM
0.35 0.4 0.45 0.5
nDCG


WT2g
TF-IDF
BM25
DFR
DRMM
QLM(dir)
NVSM
0.55 0.6 0.65 0.7 0.75 0.8 0.85
arcsin(sqrt(Precision@10))

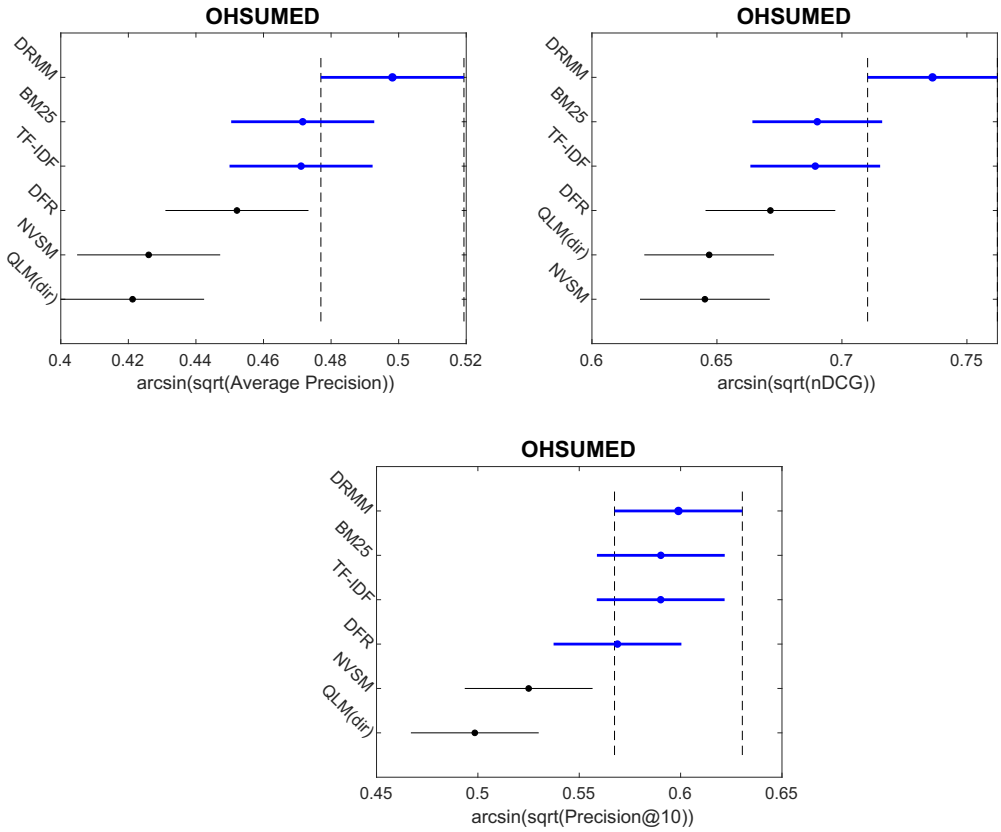


Fig. 3.3 Significance tests for the results reported in Table 3.11 and Table 3.12 (top group highlighted). All pairwise comparisons are calculated with Tukey's HSD confidence intervals and a significance level $\alpha = 0.05$. Each row depicts the comparisons made for MAP, nDCG@100 and P@10 for a specific collection.

3.12 Embedding-based evaluation

This section, evaluates the effect of different word embedding models on DRMM. We performed the evaluation on the Robust04 (T), NY (T), WT2g (TD), and OHSUMED (D) collections.

In fact, DRMM allows the usage of different word embedding models, regardless of their characteristics. Differently from NVSM, which learns its own words and documents representation, DRMM does not learn any representation. But rather, it learns to interpret the interactions between each document and query term, given a certain word representation. For this reason, we have investigated here how sensitive DRMM can be to different word embedding models.

We consider the word embedding models described in Section 5.8: (i) Word2Vec word embeddings computed with the implementation available in Gensim (W2V), (ii) FastText word embeddings obtained again with the FastText implementation from Gensim (FastText), (iii) the word embeddings obtained with NVSM (NVSM Embeddings), (iv) a set of word embeddings trained by Google on a Google News corpus. All the models, except the last one, have been trained on the experimental collections on which we later performed the retrieval experiments. We also report the performance of NVSM on each collection as a baseline.

Experimental results. From the results reported in Table 3.13, we observed the robustness of DRMM when using different embedding models. Indeed, there is no sizeable difference between the Gensim Word2Vec and the FastText embeddings on the Robust04, NY and OHSUMED collections. However, on the NY and WT2g collections FastText embeddings lead to a slightly higher performance. In general, none of the embedding models considered outperforms the others on all the collections and for all the measures. However, we observe that the Word2Vec embeddings trained by Google lead to the lowest performances overall – despite the significantly larger corpus used to train such model. Conversely, the WEs generated by NVSM outperform all the others on Robust04 and OHSUMED in MAP and P@10.

Discussion. The results in Table 3.13 suggest that DRMM has the tendency to learn to match documents leveraging more on strong exact matching signals or collection-based co-occurrence signals (NVSM WE) rather than latent semantic ones (W2V or FastText). Otherwise, DRMM would have achieved better retrieval results using a Word2Vec model trained on a very large corpus – like Google News – compared to an embedding model trained on each experimental collection. Indeed, we expect a model trained on a much larger corpus to be better at representing semantic relations between words.

	Robust04 (T)			NY (T)			WT2g (TD)			OHSUMED (D)		
	MAP	NDCG	P@10	MAP	NDCG	P@10	MAP	NDCG	P@10	MAP	NDCG	P@10
NVSM	0.139	0.269	0.279	0.141	0.211	0.274	0.225	0.380	0.402	0.214	0.335	0.319
DRMM (W2V)	0.268	**0.435**	0.455	0.141	0.211	0.274	0.289	0.455	0.434	0.272	**0.407**	0.375
DRMM (FastText)	0.262	0.427	0.456	**0.153**	**0.226**	**0.276**	**0.309**	**0.471**	**0.468**	0.268	0.402	0.368
DRMM (NVSM WE)	**0.270**	0.434	**0.458**	0.130	0.207	0.232	0.294	0.458	0.442	**0.273**	0.405	**0.397**
DRMM (W2V Google)	0.261	0.428	0.455	0.126	0.215	0.268	**0.309**	0.455	0.460	0.238	0.361	0.360

Table 3.13 Comparison between NVSM and DRMM using with different WE models: Word2Vec trained on the experimental collection (W2V), FastText trained on the experimental collection (FastText), NVSM WEs trained on the experimental collection (NVSM WE), pre-trained Word2Vec model by Google on Google News (W2V Google). The considered evaluation measures are MAP, nDCG@100 and P@10.

Interestingly, the results obtained with the embeddings generated by NVSM (NVSM WE) – trained to model term specificity along with co-occurrence relations between terms – are the best on Robust04 and OHSUMED. This result suggests that NVSM embeddings are better at modeling the collection-based co-occurrence relations between terms than Word2Vec-based approaches. This also leads to more useful matching signals that may be used to perform retrieval on the collection they are trained on. For DRMM, we can therefore conclude that matching signals based on collection-based co-occurrence relations have a more positive effect on performances than semantic ones.

Also, since there is no relevant performance difference between all the considered embedding models, we can conclude that DRMM learns to assign more importance to exact matching signals – associated to a separate bin in the histograms received in input (see Section 3.8) – than the other ones. Indeed, as shown in [39], WE models such as Word2Vec or FastText, learn very different word representations and produce different matching scores for the same terms. Hence, since the component with less variability in the matching histograms is the one associated to exact matching signals and the performance of the model is very similar for different embedding models, we conclude that DRMM learns to rely more on this dimension of the input.

3.13 Topic-based evaluation

In this section, we perform a topic-by-topic analysis of the runs of NVSM and DRMM, comparing their per-topic AP to that obtained with BM25 on the Robust04 (T), NY (T), WT2g (TD), OHSUMED (D) and CLEF (TD) collections.

We also employ *Kernel Density Estimation (KDE)* [131] to estimate the PDF of the APs of BM25, NVSM and DRMM for all topics.

Discussion. In Figures 3.4, 3.5, 3.6, 3.7, 3.8, 3.9 and 3.10 we present the scatter plots of the per-topic AP of NVSM, DRMM and BM25 on different collections. For each collection, we compare each model with the others.

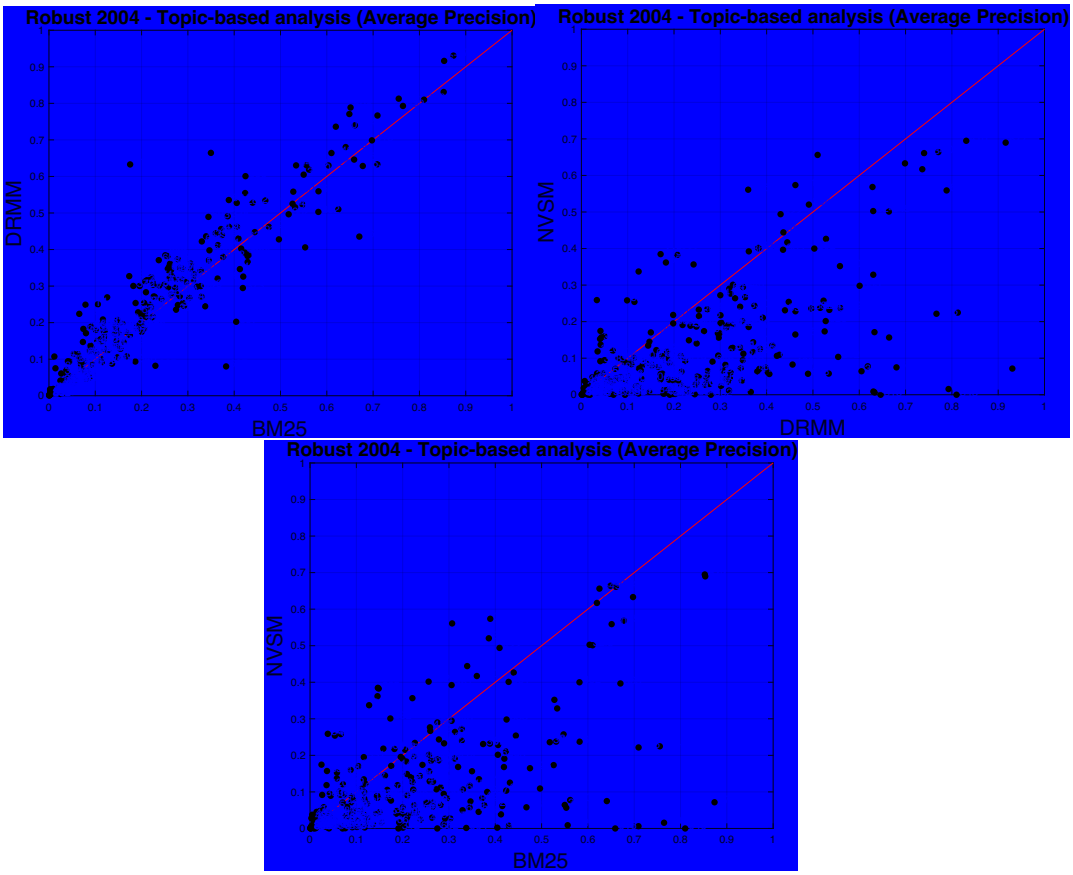


Fig. 3.4 Scatter plots of the AP for each topic of the Robust04 collection, obtained with DRMM, NVSM and BM25 retrieval models.

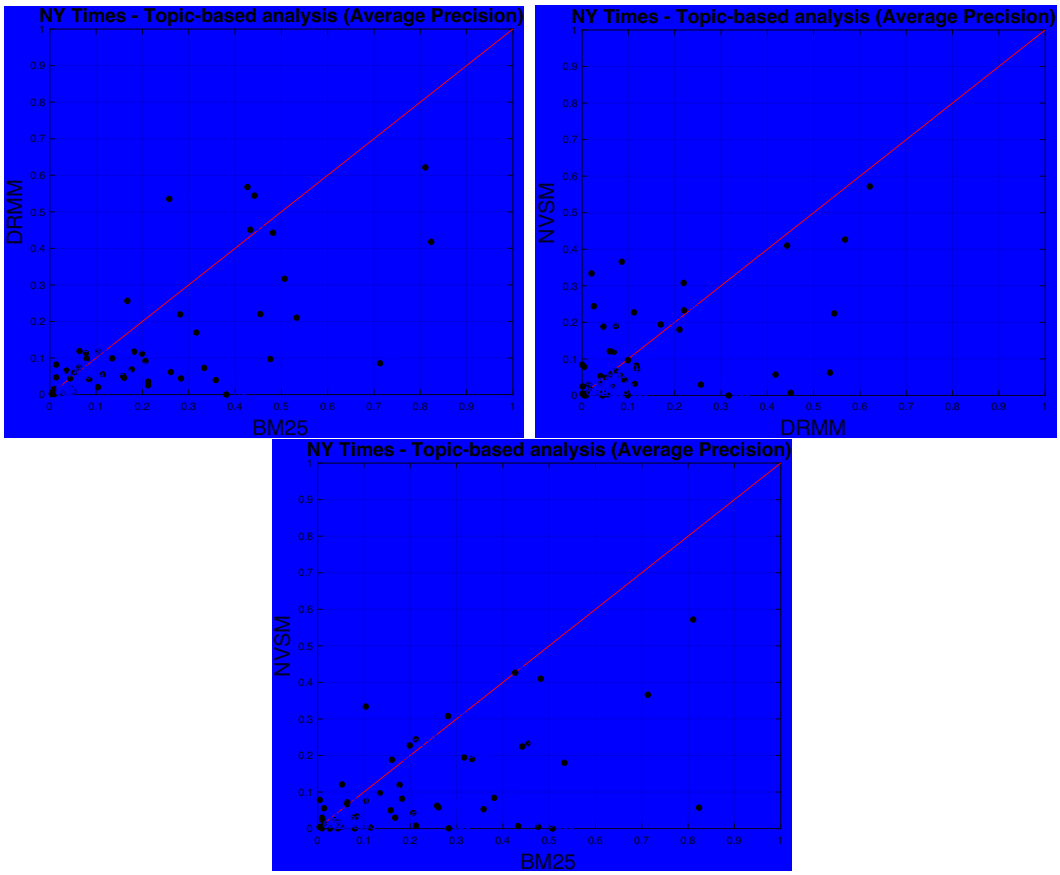


Fig. 3.5 Scatter plots of the AP for each topic of the NY collection, obtained with DRMM, NVSM and BM25 retrieval models.

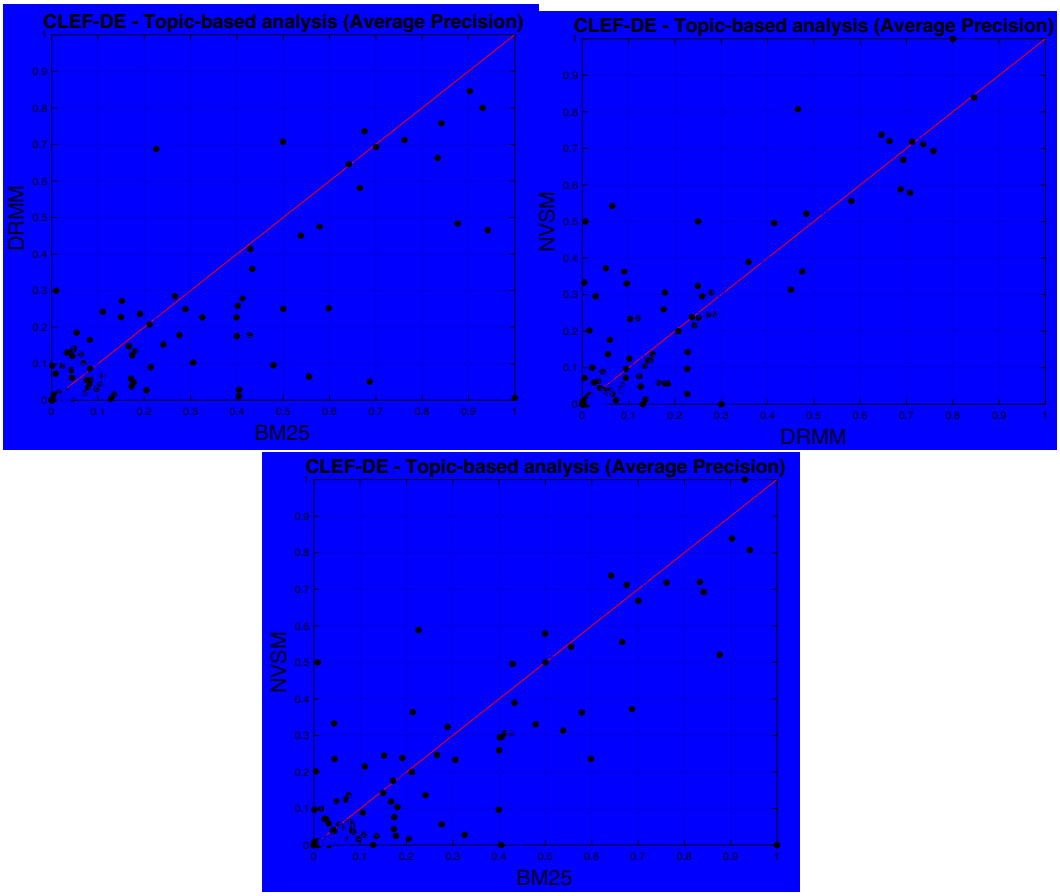


Fig. 3.6 Scatter plots of the AP for each topic of the CLEF-DE collection, obtained with DRMM, NVSM and BM25 retrieval models.

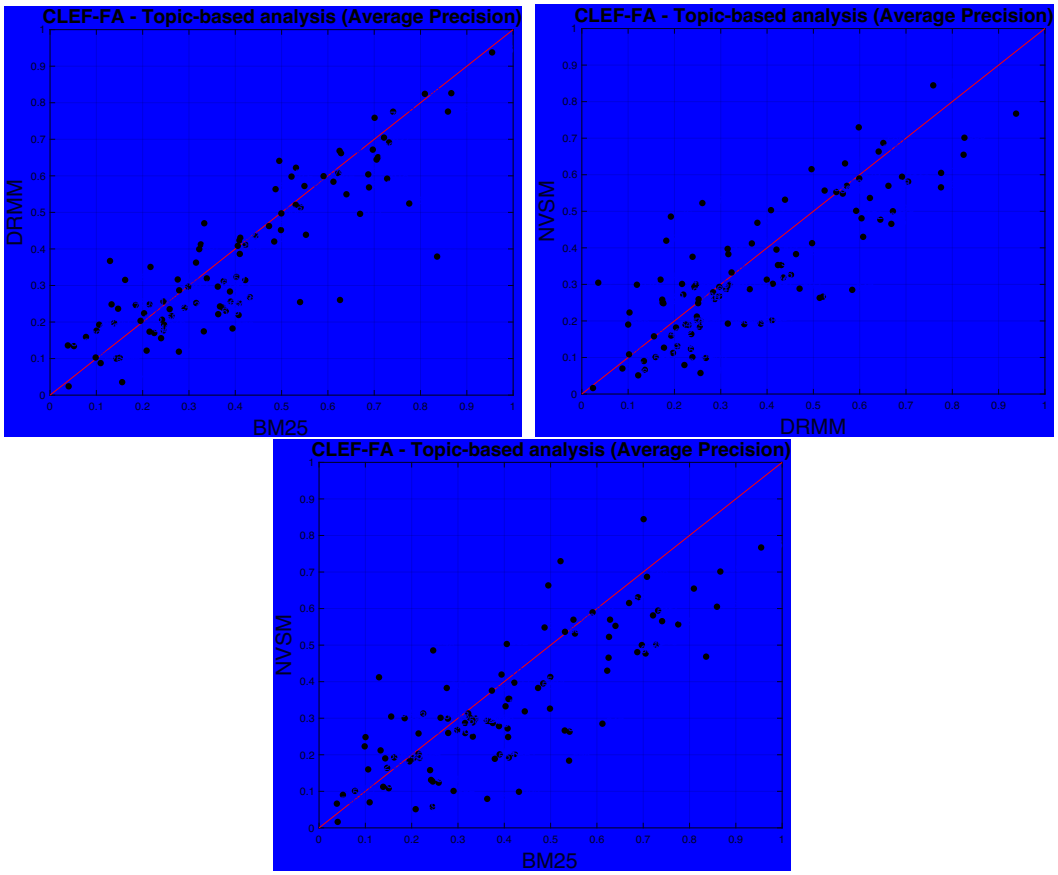


Fig. 3.7 Scatter plots of the AP for each topic of the CLEF-FA collection, obtained with DRMM, NVSM and BM25 retrieval models.

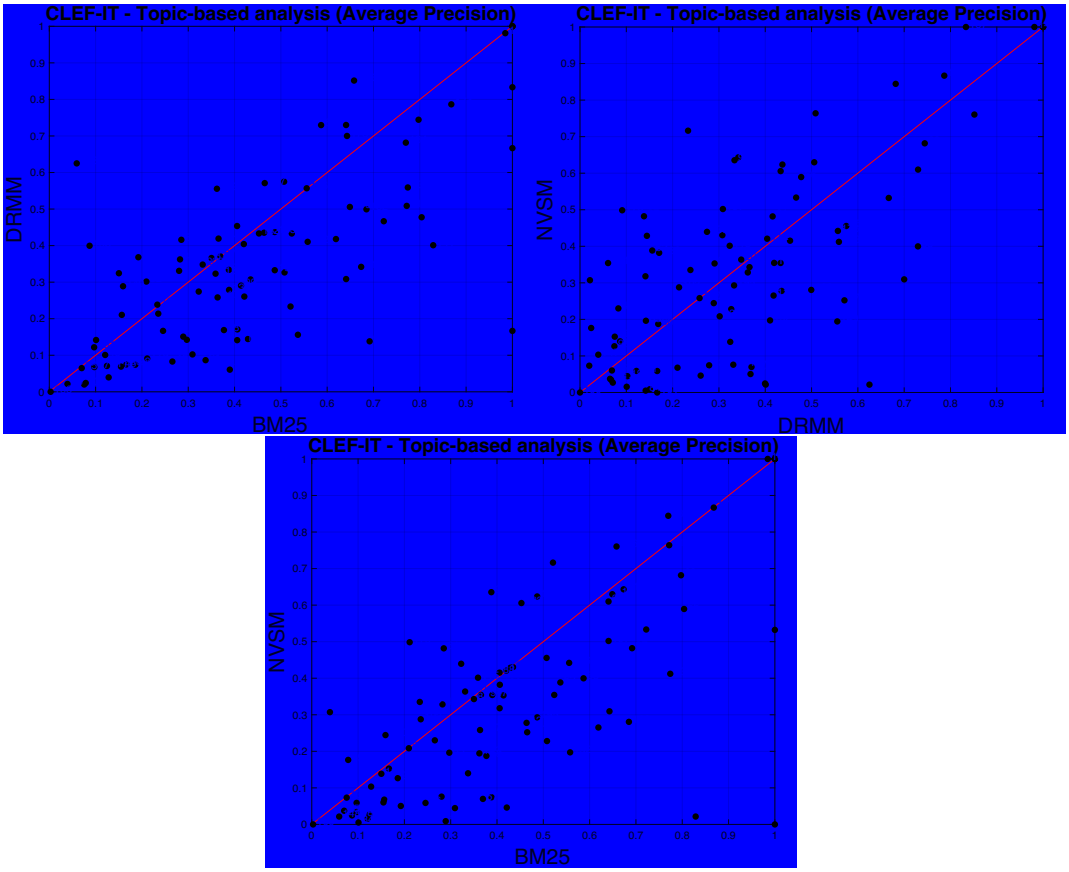


Fig. 3.8 Scatter plots of the AP for each topic of the CLEF-IT collection, obtained with DRMM, NVSM and BM25 retrieval models.

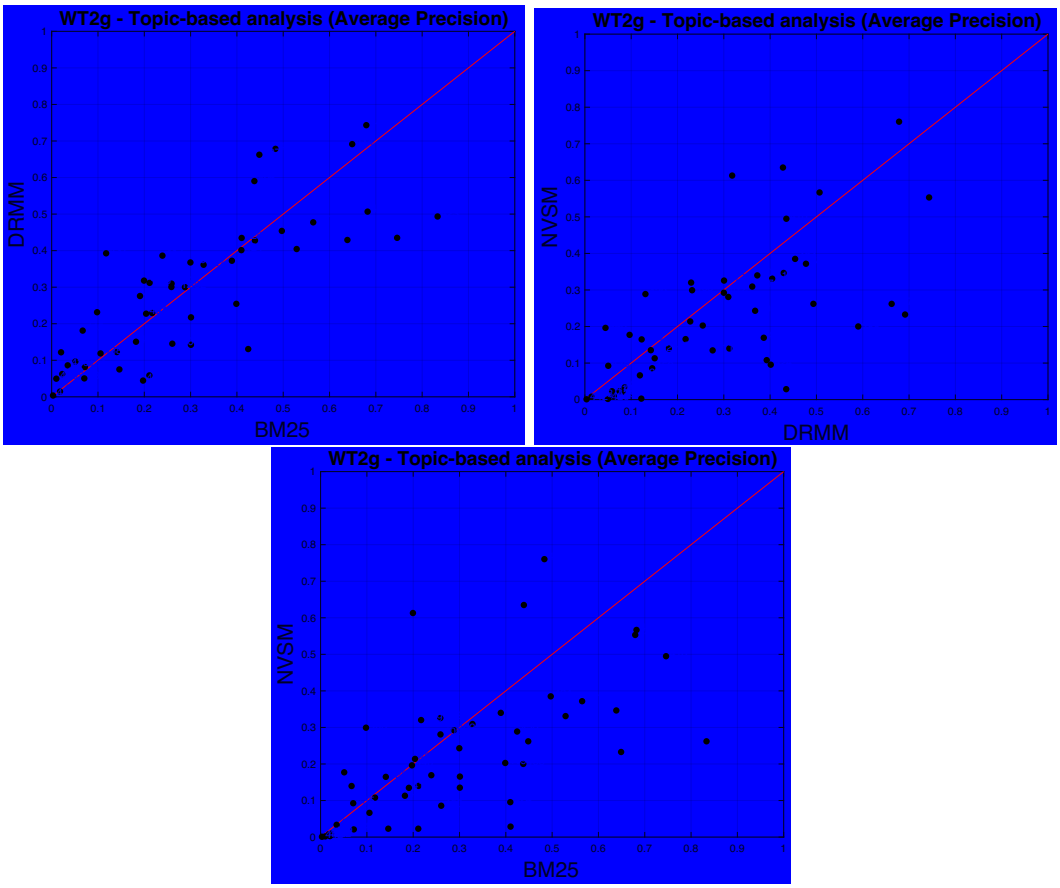


Fig. 3.9 Scatter plots of the AP for each topic of the WT2g collection, obtained with DRMM, NVSM and BM25 retrieval models.

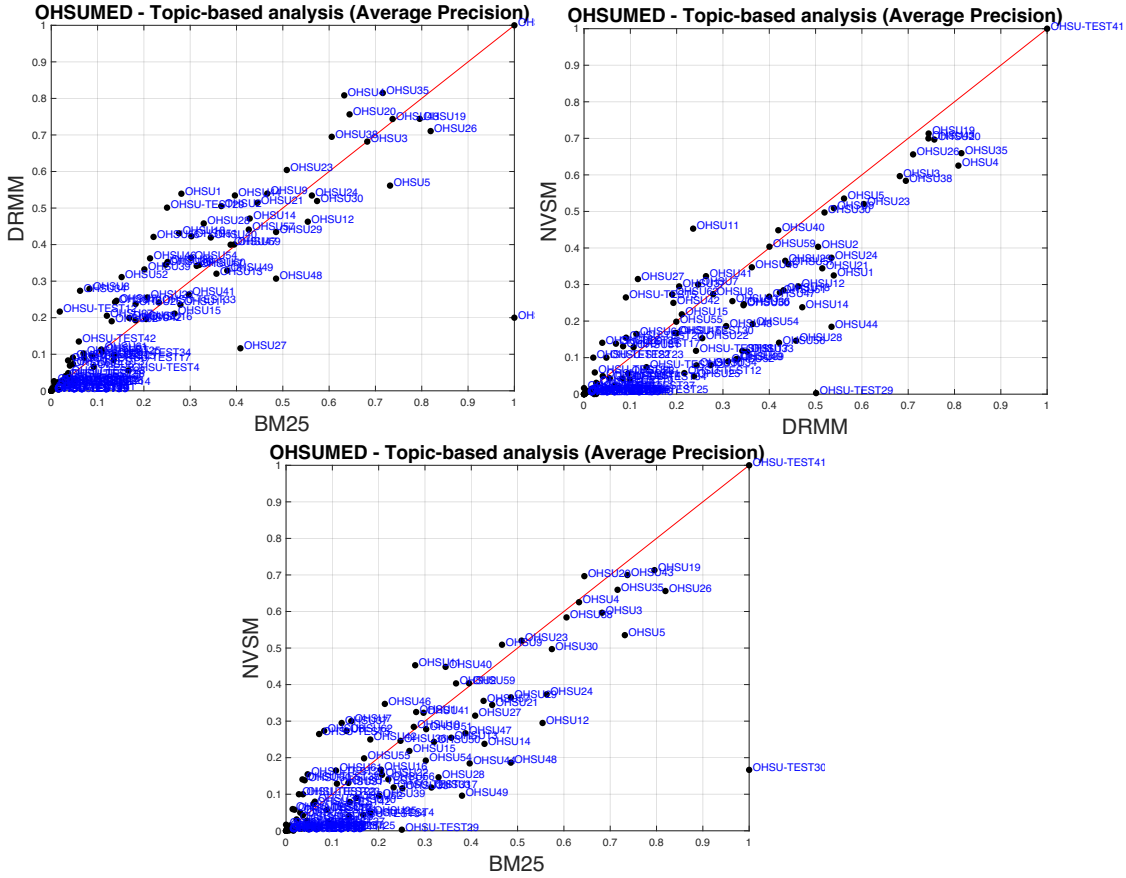


Fig. 3.10 Scatter plots of the AP for each topic of the OHSUMED collection, obtained with DRMM, NVSM and BM25 retrieval models.

We begin to consider the results on the Robust04 collection. If we compare DRMM and BM25, we observe that most of the points are close to the bisector. This indicates that, in general, the two models have a similar performance on most of the topics. On the other hand, the comparison between NVSM and BM25, reveals that the majority of the points is concentrated below the bisector, meaning that BM25 outperforms NVSM on most of the topics. If we analyze the results of the comparison between NVSM and DRMM, we observe a similar point distribution as in the previous chart, once again indicating, on this collection, DRMM's greater effectiveness over NVSM.

Considering the results on NY, DRMM reveals a worse performance than BM25 on most of the topics, and so does NVSM. However, in comparing the two NIR models, we see that DRMM outperforms NVSM on a large number of topics. The large performance difference between DRMM and BM25 is likely due to the fact that the NY collection only has 50 topics, which are not enough for DRMM to learn a good ranking model as done in other collections. However, since DRMM re-ranks on a set of 2000 documents previously retrieved with QLM (dir), it has an advantage over NVSM which is also confirmed by its performance difference. Examining the documents retrieved by NVSM in greater detail, and comparing them with the ones returned by BM25, we noted that NVSM can return relevant documents which do not contain any query term. For instance, on topic 442 ("heroic acts") NVSM returns a relevant document – with doc id "1036498" – which does not contain any query term and is not returned by BM25. In this case, terms like "heroism" and "sacrifices" are used in the documents instead of those of the query. Similar situations for other topics are observable in other collections (i.e. OHSUMED).

On CLEF collections, we observe similar relative performances as in the two previous collections. DRMM performed overall similarly to BM25 except for a few topics where it outperforms it, i.e., topic 200 in CLEF-DE, and topic 148 in CLEF-IT; or vice versa, where BM25 performs better by a large margin, i.e., topics 141, 149, and 161 on CLEF-DE, topic 628 on CLEF-FA and topics 161 and 44 on CLEF-IT. On the other hand, NVSM is in general outperformed by BM25 on all the collections for most of the topics. This time however, the difference is less marked than on the Robust04 and NY collections. Finally, if we compare DRMM and NVSM on the same set of collections, we observe a performance difference of the two models for a large number of topics mostly on the CLEF-IT collection – where the points in the scatterplot are in general further from the bisector – while we observe a similar performance on the other two collections in the majority of the topics with just a few outliers where NVSM outperforms DRMM by a larger margin on CLEF-DE.

On the WT2g collection, we observe that DRMM is performing once again in the majority of the topics in a very similar way to BM25 – with the exception of topics 423 and 410 where

BM25 outperforms DRMM by a large margin. NVSM on the other hand is outperformed by the two other models on the vast majority of the topics. However, we also observe that in a handful of topics, DRMM and NVSM both manage to outperform BM25. This happens for topic 416: "Three Gorges Project What is the status of The Three Gorges Project?". In this case, the documents containing the topic keywords are very long and BM25 does not recognize many of them as relevant.

Finally, on OHSUMED, we observe a similar relative performance of the models as on the Robust04 collection. Where DRMM is performing on all topics in a similar way as BM25, and NVSM is in general outperformed by the two other models in most of the topics. Here, as in the NY collection, there are some topics, for which NVSM returns relevant documents which do not contain any query term. This is the case for topic OHSU7 ("lactase deficiency therapy options") and document with doc id "91359745" which contains none of the query terms but only their synonyms or closely related terms (i.e. "lactose" and "intolerance").

Overall, we can conclude that DRMM and BM25 have a very similar performance across all topics, with a few outliers only on CLEF-DE and NY, while NVSM is usually outperformed by both BM25 and DRMM on most of the topics and collections. If we consider the differences between NVSM and the two other models we also observe that these are always larger than DRMM and BM25. This is likely due to the fact that DRMM performs a re-ranking on the top 2000 documents retrieved with QLM (dir) – a lexical model like BM25. For this reason, we expect the rankings of NVSM to contain a more diverse set of documents than the ones in the runs produced by DRMM or BM25 since NVSM relies solely on the neural model without explicitly considering exact term matches.

Considering the charts in Figure 3.11, we can assess the differences in the systems' behavior that is otherwise not identifiable when observing the average measures across all topics. In this case, we see that on the Robust04, NY, CLEF-DE, WT2g and OHSUMED collections, the MAP value is highly influenced by the low performances of the systems on a large number of topics – this situation is especially true for NVSM. In fact, in all these charts there is a high peak associated to low AP values. Conversely, the experiments on the CLEF-IT, CLEF-FA collections, highlight that the final MAP value is dominated by a large number of topics where the systems obtain an AP around 0.3 – except for NVSM which also has a high peak close to 0.1.

The quantitative distance in the distribution of AP values across different topics can be measured considering the KLD between the AP distributions reported in Figure 3.11. Table 3.14 illustrates the distances between the AP distribution associated to the BM25, NVSM and DRMM models.

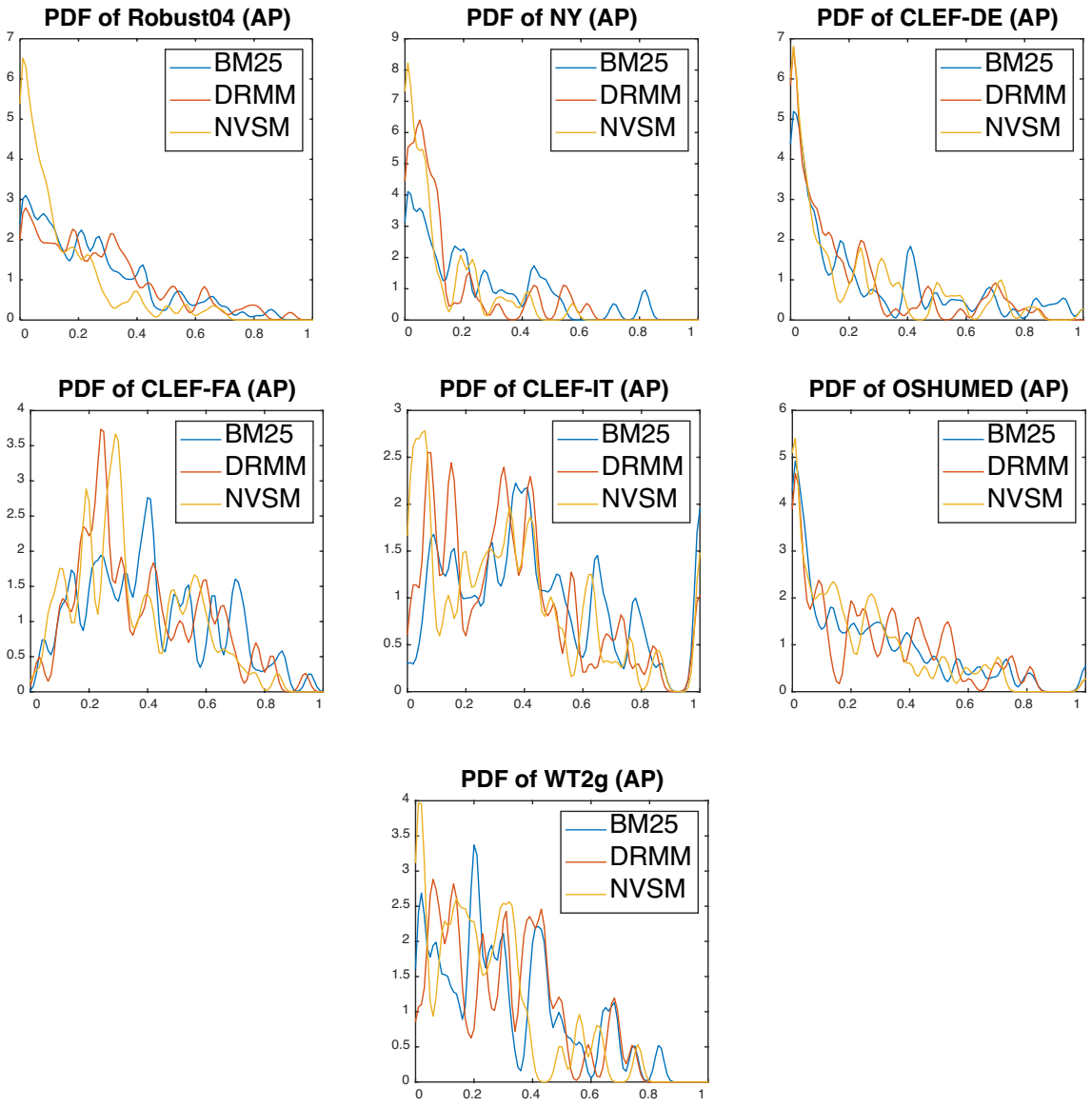


Fig. 3.11 PDF of the AP relative to WT2g, OHSUMED, CLEF-IT, CLEF-DE, and CLEF-FA collections, of NVSM and BM25 retrieval models. Note that on the x-axis there are the AP values distributed into 100 bins (from 0 to 1 with 0.01 step) and on the y-axis the density estimation.

	Robust04	NY	CLEF-DE	CLEF-FA	CLEF-IT	OHSUMED	WT2g
BM25 - DRMM	10.86	63.29	21.76	18.58	14.48	16.91	27.08
BM25 - NVSM	20.57	40.69	28.15	22.70	26.65	49.60	61.22
DRMM - NVSM	31.52	23.85	24.06	16.45	18.69	71.66	76.29

Table 3.14 KLD values between the PDF of the AP values obtained with BM25, NVSM and DRMM on the Robust04, NY, CLEF-DE, CLEF-FA, CLEF-IT, OHSUMED, and WT2g collections. KLD $\in [0, +\infty)$, it denotes the divergence between the two distributions [144]; therefore, 0 means that the two models behave in the same way for all the topics in the collection; $+\infty$ means that for no topic the two models behave in the same way.

Considering the distances among the distributions on the Robust04, CLEF-DE, CLEF-IT, OHSUMED and WT2g collections, we notice that DRMM and BM25 are often the closest systems. This indicates that DRMM and BM25 have a comparable number of topics where they obtain a similar AP. Worth noting that the set of topics with a similar AP might be different since we are only considering the distribution of the AP values, ignoring any corresponding topic ids. On the NY and CLEF-FA collections instead, DRMM and NVSM are the two closest systems, implying that they have a similar number of topics with comparable AP values. This also corresponds to the results reported in Figure 3.11, where both models show very high peaks associated to AP values around 0.1 and 0.2 on the NY and CLEF-FA collections, respectively.

In conclusion, the main advantage observed of the considered NIR models is the ability to retrieve documents which do not contain any query term. Moreover, in a few cases we observed that NIR models have the ability to rank higher than lexical models very long documents containing relatively short relevant passages related to a query. Nevertheless, we did not find enough empirical evidence to strongly support this claim.

Finally, NIR systems do not outperform in general traditional lexical approaches such as BM25. It can however be noticed that DRMM is competitive with lexical models on certain collections (Robust04, WT2g and OHSUMED), and outperforms BM25 on a few topics.

3.14 Final Remarks

In this chapter, we presented an in depth analysis of two paradigmatic NIR models, i.e. DRMM and NVSM, through a reproducibility study. The neural models we selected are prominent examples of the NIR wave and they report competitive results for ad-hoc retrieval. DRMM is a supervised approach which performs a re-ranking of the documents retrieved by another IR model. This approach relies on a set of pre-trained WEs – obtained with Word2Vec – to extract semantic matching signals which, in turn, are used to perform the re-

ranking of documents. NVSM, on the other hand, is an unsupervised model which performs retrieval on the whole collection instead of reranking.

Overall, we observed that the performance of these NIR systems can often be competitive – depending on the topic and the domain of application – with traditional lexical retrieval models. The quality of their output rankings is also often tightly linked with the collection pre-processing steps and the word representation model used, e.g. Word2Vec or FastText. Indeed, we observed that domain and topic-specific word representations often yield a significant advantage to the NIR model using them.

Through a topic-level analysis of the results, we also observed how DRMM and NVSM can significantly outperform lexical state-of-the-art models such as BM25. This evaluation strategy offered us the opportunity to provide a more variegated analysis compared to averaged statistics that are more often employed for performance evaluation in the majority of conference papers. More specifically, our analyses showed how DRMM is better capable than BM25 of evaluating relevant documents where query terms appear with a lower frequency. On the other hand, through semantic matching and bypassing the reranking stage, NVSM can retrieve documents which do not contain any query term at all. This characteristic is however insufficient to outperform BM25 on average. In other words, when semantic matching is required, or there is the need to focus on a limited portion of the document containing relevant information, NIR models tend to outperform lexical ones. However, in the considered collections the number of topics where these characteristics are needed is quite limited. As a consequence, the effect of semantic matching has a minor impact on average performances.

Following of our experimental evaluation, we believe that to truly outperform traditional lexical models, NIR approaches should not just rely on an accurate and domain-aware representation of words, but should also improve their capacity to assess the relevance of different parts of a document, depending on the user query.

Until then however, we believe focusing on developing better ways of integrating NIR systems in traditional retrieval pipelines could yield significant performance improvements in the short term. For example, based on our experimental results, employing topic-aware model to select the best retrieval and reranking pipeline for a specific user query could already improve the quality of an IR system response by deciding whether to employ a neural reranking strategy or a fully neural retrieval pipeline instead of a lexical retrieval model like BM25 and vice versa.

Chapter 4

Probabilistic Word Representations

4.1 Introduction

One of the most prominent characteristics that make NIR models stand out from other retrieval and ranking approaches is their ability to automatically extract relevance patterns from raw text [35, 8] by comparing query and document terms. As shown in Chapter 3, differently from lexical retrieval approaches, NIR models can retrieve relevant documents which do not contain any query term. The development of WE played a crucial role in this methodological advancement. Indeed, WE have already been widely used in NIR [8] for the generation of semantic matching signals – which can be further interpreted by a neural model to return query-document relevance scores – and much work has been done in the past few years in the IR community on the study and development of new neural architectures to interpret them [8]. However, less attention has been devoted to studying or improving the representation of words being used – at least within the IR community. Within this research area, most approaches rely on text representations obtained with Word2Vec [145], GloVe [146], and BERT [7], or learn their own model-specific word representations like NVSM [13] and DSSM [17]. Some other works also considered the possibility of learning WE specific for relevance matching [147, 148]. Nonetheless, the number of approaches adopting the latest strategies for natural language representation remains limited.

For this reason, we evaluate in this chapter the impact of probabilistic representations of words, i.e. Elliptical *Probabilistic Word Embeddings (PWE)* [36] to the IR task.

4.2 Research Question

Elliptical PWE are a recent development within the broader PWE domain [149] that includes approaches to model complex word patterns in text by introducing "uncertainty" in embedded representations of words. Their aim is to generate semantically richer matching signals to be used for similarity matching. Indeed, point embeddings, such as those obtained with Word2Vec, can be regarded as a particular (and degenerate) case of PWE where "the uncertainty is infinitely concentrated on a single point" [36]. Probability similarity measures based on PWE are broader and provide an opportunity for additional flexibility in the definition of similarity between words. The main advantage of PWE is the ability to encode more complex word relations such as hypernymy [36] – a word A is said to be a hypernym of a word B if any B is a type of A, for example any dog (B) is a type of mammal (A). This may have meaningful implications in IR, where this characteristic would help to extract a wider range of similarity matching signals between query and document terms. In particular, this strategy goes in the direction of filling the aforementioned semantic gap of IR, i.e. the problem of extracting meaningful matching signals between terms which are semantically related but have a different spelling. This problem is especially felt in ad-hoc retrieval where we expect to receive a large variety of user queries, sometimes referring to the same concepts expressed in different ways. We believe PWE could have an advantage in this context compared to traditional WE models thanks to their richer representation model. For these reasons, in this chapter we will be focusing on answering the following research question.

> **RQ2:** Can probabilistic word representations improve the performance of NIR systems?

4.3 Methods

To conduct our analysis on the impact of PWE on NIR systems, we first perform a qualitative evaluation of these embeddings. Then, we adapt an existing NIR architecture – i.e. MatchPyramid [35] – to use this new representation of words and evaluate its performance when using PWE. To better evaluate the advantage of encoding broader relations between terms in WE, we also compare the trained PWE to a similar type of word vectors, *WordNet Embeddings (WNE)* [37]. This representation is not obtained from raw text data like Word2Vec embeddings or PWE, but from a graph reproducing the relations between a selection of words such as WordNet. These relations are therefore encoded in the WE, obtaining a set of vectors with similar properties to PWE. These embeddings, however, do not perform well in the retrieval task since the set of words and relations which can be encoded is limited by the

size of the ontology adopted – this leads to a marked *Out-Of-Vocabulary (OOV)* problem, i.e. the issue that occurs when we need to find a representation for a term which is not available in the pre-computed WE model we are using. To address this issue, we propose an extension of the MatchPyramid architecture that combines WNE matching signals with those obtained from character n-grams embeddings pre-trained with fastText [150]. The broader purpose of this analysis is to understand the potential contribution of complex and more flexible WE in IR. Hence, we consider MatchPyramid trained on traditional Word2Vec embeddings as our baseline and we compare our extended architectures against it.

4.4 Contributions

Our contributions regarding the aforementioned research question are summarized below.

- We provide a qualitative analysis of Elliptical PWE, comparing them to other popular WE models such as Word2Vec, fastText and WordNet embeddings. With our evaluation, we confirm how this type of WE has an advantage over other models in representing the latent relations between the different meanings of words;
- Next, we provide a quantitative evaluation of the impact of the aforementioned WE models on the retrieval performance of the MatchPyramid model. Our results show the potential of PWE in the IR ad-hoc retrieval task;
- Finally, we conduct a topic-level evaluation of the impact of PWE on the retrieval performance of the same MatchPyramid model. Our evaluations show how PWE lead to more widespread performance improvements compared to other WE models.

4.5 Outline

This chapter is organized as follows. In Section 4.6, we present some background information on WE with a focus on NIR; in Sections 4.7 and 4.8, we introduce PWE and WNE; in Section 4.9 we present our experimental setup; in Section 4.10, a comparison between PWE, *fastText Embeddings (FTE)* and WNE, and their evaluation on the retrieval task; in Section 4.11 we present our final remarks on this chapter.

4.6 Related Work

Neural models have been employed in IR to automatically detect regularities in matching patterns and exploit them for document retrieval. These models usually rely on WE that can be pre-trained or learned from scratch to enable semantic matching between query and document terms and consequently improve retrieval [8]. Over the years, the IR research community adopted a wide variety of WE models, from Word2Vec to the most recent BERT. Here, we will focus on non-contextual representations of words, also known as BOW embedding models, i.e. WE models where the vector representation associated to each term is independent from the context where it is used. These models offer numerous advantages in terms of efficiency compared to contextual WE models because they allow to reuse the same word representation regardless of its context. On the other hand, contextual WE require a series of operations to obtain the vector representation of a new term, whenever its context changes. This limits the application or highly increases the cost of employing these models in modern search systems requiring a short response time. For this reason, we are going to focus in this analysis on BOW models like Word2Vec, GloVe, fastText, WordNet embeddings and PWE.

Point word embeddings. Point word embeddings such as Word2Vec [145] and GloVe [146] have been widely used in NIR systems because they allow to incorporate a mixture of topical and typical notions of relatedness between terms in their representations, leveraging only on their co-occurrences in large collections of documents. fastText [150] is an extension of Word2Vec to compute word representations based on character n-grams. Using subword-level information is particularly interesting to build vectors for unknown words. This is done by summing the character n-grams in the unknown term. For instance, the tri-grams in the word “apple” are “app”, “ppl”, and “ple” (ignoring the starting and ending of boundaries of words). Hence, the word embedding for “apple” will be the sum of these n-gram vectors. However, since these models are trained to encode only co-occurrence relations, they have some drawbacks when used in NIR. In co-occurrence based models, word embeddings of terms which are often used in the same contexts have a high similarity score despite having different meanings (e.g. the terms “dangerous” and “safe”) [147] and this could lead to misleading matching signals for NIR models. A solution to this problem is to employ contextual word representations of words such as BERT [27] or XLNET [151]. However, despite achieving state-of-the-art performance on lots of NLP and IR tasks, these models still remain impractical and too expensive to employ in many real-world applications.

WordNet-based WE. WordNet [152] [1] is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. WordNet has been widely used in IR, but was never competitive with state-of-the-art approaches [153–156]. The main issues that need to be faced when WordNet is employed by an IR system – e.g., for query or document expansion [157–159] – are the lack of collection-specific terms in the database, the need to design a customized similarity measure among terms [160] or the difficulty of extracting the correct word sense from the ontology. With the diffusion of word embeddings, new approaches were also developed to employ this representation with lexical resources such as WordNet, Wiktionary [2] or Wikidata [3] [161, 37]. A vector representation of words has the advantage of making the computation of the similarity between terms much faster than when using a graph-based distance metric such as Wu and Palmer similarity [160]. Here, we employ WordNet to create word representations which do not present the same drawbacks in NIR of co-occurrence based models such as Word2Vec, and encode richer relations between words such as hypernymy.

PWE. Typically, WE are obtained by embedding each word into a point/vector in the Euclidean space $\mathbb{R}^d$, where a distance-based similarity measure (e.g., cosine similarity) reflects the semantic similarity [145]. While this approach has been proven effective in many NLP and IR applications, recent trends started to generalize point embeddings using probability measures spaces. In this case, each word is represented by a mean vector and covariance matrix pair introducing uncertainty in the representations. The main advantage of such approach is the ability to encode the different senses of a word or complex word relations such as hypernymy through a covariance matrix [149, 36]. In the literature, various methods to train probabilistic embeddings have been proposed, starting from [149] which pioneered the method introducing Gaussian Embeddings where each word is represented by a Multivariate Gaussian distribution. The similarity metrics employed are the Probability Product Kernel or *Kullback-Leibler Divergence (KLD)*. Nevertheless, these metrics have some drawbacks since they diverge for degenerate covariance matrices and assume high values when the variance is very small. These problems can be settled by working on the space of elliptical distributions (i.e., probability measures with elliptical level sets) endowed with the Wasserstein-Bures product [36]. Indeed, the Wasserstein-Bures product handles degenerate cases and has better numerical stability than the metrics employed in [149]. Moreover, elliptical embeddings are flexible since they can be transformed into Gaussian

[1] `https://wordnet.princeton.edu/`
[2] `https://www.wiktionary.org/`
[3] `https://www.wikidata.org/`

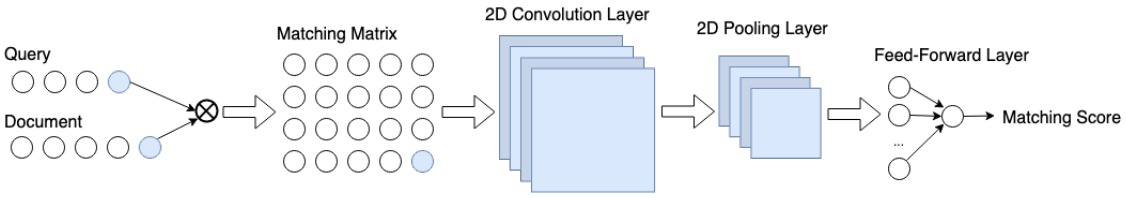


Fig. 4.1 Architecture of MatchPyramid.

Embeddings or point embeddings by assuming the probability measures to be a Gaussian or a Dirac delta, respectively. Finally, elliptical embeddings are stable and provide interesting numerical properties that guarantee smooth training and easy employment with neural architectures; for these reasons, we prefer them to other recent developments in probabilistic embeddings, such as box embeddings [162].

The MatchPyramid model. Here, we evaluate different WE – with a focus on PWE – and study how they impact the performance of a NIR system. The NIR model we employ is based on the MatchPyramid architecture [35]. We selected this architecture because, despite being a supervised deep neural model, it does not require lots of training data [35] and can be applied to relatively small IR collections such as TREC Robust04 [140]. Moreover, it relies solely on the interactions between query and document terms, and processes these raw matching signals directly in its neural model without performing any prior operation on them. This enables a more accurate evaluation of the impact of the new WE, than other NIR models – possibly also better performing, such as DRMM [15] – which alter the matching signals before processing them. Finally, MatchPyramid is a better performing approach than similar deep neural architectures for text matching such as DeepMatch [18] or other approaches based on Convolutional Neural Network (CNN) such as ARC-I/ARC-II [16]. The architecture of MatchPyramid is depicted in Figure 4.1. First, the model receives as input two embedded sequences of words, a query and a document. Second, it computes a similarity matrix between each pair of terms from the query and the document. In our experiments, we used cosine similarity which, despite not being the best performing one for the model [35], is the one used in Word2Vec and fastText for the training of the embeddings. Cosine similarity is the best metric to evaluate the WE characteristics since the relative positions of the embeddings in the space have been optimized during training according to this distance. Third, a 2D-CNN is applied to the matching matrix to extract local matching signals, followed by a dynamic pooling layer [14]. Finally, a shallow Feed-Forward (FF) neural network with two layers is used to compute the matching score between the query and

the document in input. The model was trained to minimize the loss function:

$$L(q,d^+,d^-;\theta) = max(0,1-S(q,d^+)+S(q,d^-)), \tag{4.1}$$

where θ indicates the model parameters, q represents a user query and d^+, d^- are respectively a relevant and a non-relevant document for q. To decrease retrieval time, we filtered the documents to be ranked with MatchPyramid, computing the matching score for each query only for the top 2K documents retrieved with the QLM model [45] in Terrier v.4.1. [4]

4.7 Elliptical Probabilistic Embeddings

PWE generalize classical word embeddings considering each word representation as a probability measure. In the case of elliptical distributions, a measure $\mu_{\boldsymbol{m},\boldsymbol{M}}$ is determined by its mean ($\boldsymbol{m} \in \mathbb{R}^d$) and covariance matrix ($\boldsymbol{M} \in \mathbb{R}^{d\times d}$). Since the covariance matrix is symmetric, the number of parameters required for each embedding can be computed as $d + \frac{d(d+1)}{2}$, where d is the size of the vector $\boldsymbol{m}$. Let $\mu_{\boldsymbol{a},\boldsymbol{A}}$ and $\mu_{\boldsymbol{b},\boldsymbol{B}}$ be two elliptical measures, we define the Wasserstein-Bures pseudo-dot-product [36] as:

$$[\mu_{\boldsymbol{a},\boldsymbol{A}} : \mu_{\boldsymbol{b},\boldsymbol{B}}] = \langle \boldsymbol{a},\boldsymbol{b}\rangle + Tr\left(\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{B}\boldsymbol{A}^{\frac{1}{2}}\right)^{\frac{1}{2}} \tag{4.2}$$

Here $\langle\cdot,\cdot\rangle$ is the scalar product and $Tr(\cdot)$ is the trace operation. Such pseudo-dot-product can be seen as an extension of the standard dot product to the space of elliptical measures and defines a similarity measure between them. This allows us to train the embeddings using i.e. the Continuous Bag of Words (CBOW) [145] paradigm, maximizing the similarity between the target word and the words in its context window while minimizing it for words outside the context. For a word w, its context is defined as the N_w^+ words located before and after each occurrence of w in the corpus; the rationale behind this training strategy is that words surrounded by similar contexts should be semantically correlated. Formally, this translates into minimizing the following hinge loss (Eq. 4.3) on the training set elements:

$$\sum_{w\in\mathscr{W}}\left[\mathscr{M} - \frac{1}{N_w^+}\sum_{c^+\in\mathscr{C}_w^+}[\mu^{c^+}:\mu^w] + \frac{1}{N_w^-}\sum_{c^-\in\mathscr{C}_w^-}[\mu^{c^-}:\mu^w]\right]. \tag{4.3}$$

For each occurrence of the word $w \in \mathscr{W}$ in the corpus, we indicate with $\mathscr{C}_w^+$ the sets of terms of size N_w^+ surrounding it (positive context), and with $\mathscr{C}_w^-$, the negative samples

[4] The modified version of Terrier we used for our experiments is available at the url: `https://github.com/gridofpoints`

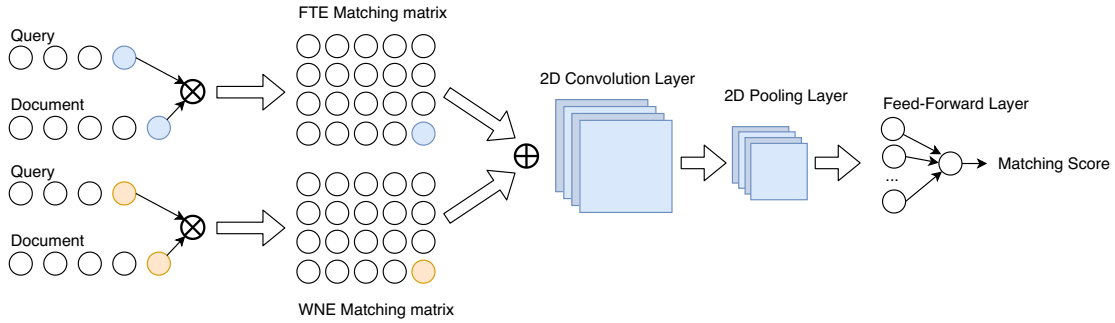


Fig. 4.2 Architecture of MatchPyramid with WNE.

(i.e. randomly sampled words not belonging to the current context of w) of cardinality N_w^-. Moreover, we indicate with $\mu^w, \mu^{c^+}, \mu^{c^-}$ respectively the embeddings of w, c^+ and c^- parametrized by their mean and covariance matrices – we do not indicate them here in order to simplify the notation – and $\mathscr{M}$ is a slack variable. An important aspect of the training procedure is that the covariance matrices are required to be symmetric positive definite. In order to satisfy this requirement, we factorize them as $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$, learning $\boldsymbol{L}$ during the optimization. For more details on optimization and differentiation of the loss function we invite the readers who are interested to refer to [36]. The dot product defined in Eq. 4.2 is affected by the word frequencies in the training corpus; hence, it is necessary to define its normalized counterpart to be used during the word similarity evaluation phase. Given two words w_a and w_b and their elliptical measures $\mu_{\boldsymbol{a},\boldsymbol{A}}$ and $\mu_{\boldsymbol{b},\boldsymbol{B}}$, we define the normalized dot product as:

$$\mathscr{C}[\mu_{\boldsymbol{a},\boldsymbol{A}} : \mu_{\boldsymbol{b},\boldsymbol{B}}] = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{||\boldsymbol{a}|| \cdot ||\boldsymbol{b}||} + \frac{Tr\left(\boldsymbol{A}^{\frac{1}{2}}\boldsymbol{B}\boldsymbol{A}^{\frac{1}{2}}\right)^{\frac{1}{2}}}{\sqrt{Tr\boldsymbol{A} \cdot Tr\boldsymbol{B}}} \tag{4.4}$$

$\mathscr{C}[\mu_{\boldsymbol{a},\boldsymbol{A}} : \mu_{\boldsymbol{b},\boldsymbol{B}}]$ has its maximum value 2 when $\mu_{\boldsymbol{a},\boldsymbol{A}} = \mu_{\boldsymbol{b},\boldsymbol{B}}$ and defines the similarity between w_a and w_b in an elliptical space.

We were able to employ PWE in the NIR MatchPyramid model, by changing the way the matching matrix between query and document terms is created. In this case, instead of cosine similarity, we employed the Wasserstein-Bures normalized dot product as a similarity measure between two terms.

4.8 WordNet Embeddings

Classical word embedding models such as Word2Vec, GloVe or fastText, employ large collections of documents to train word representations. These approaches leverage on co-occurrence relations between words in order to encode semantic information in their vector

representation. On the other hand, the goal of WordNet embeddings is to employ semantic networks (i.e. WordNet) to infer word representations. In this case, WE encode the same information of the semantic graph used to create them, but in a more compact way. This information is then used in neural systems with more efficiency and little modifications to existing architectures. A semantic network is a graph where each node represents a term and each edge indicates a relation between two terms. The strength of the semantic affinity of two words is proportional to the number of paths connecting two nodes – the higher the number of paths, the stronger the affinity – and to their length – the shorter the paths, the stronger the affinity.

WordNet Embeddings (WNE) are computed as follows. First, the semantic network G is represented with an adjacency matrix M where $M_{ij} = 1$ if words w_i, w_j are connected by an edge in G, and 0 otherwise. Second, in order to also represent weaker relations between terms, the following operation is applied:

$$M_G^{(n)} = I + \alpha M + \alpha^2 M^2 + ... + \alpha^n M^n, \tag{4.5}$$

where I is the identity matrix, M^n is the matrix where $(M_{ij})^n$ counts the number of paths of length n between nodes i, j in G, and $\alpha < 1$ is a decay factor determining how longer paths are dominated by shorter ones. This iterative procedure is repeated until $M_G^{(n)}$ converges into matrix M_G, which is analytically obtained by an inverse matrix operation given by [37]:

$$M_G = \sum_{e=0}^{\infty} (\alpha M)^e = (I - \alpha M)^{-1}. \tag{4.6}$$

Third, each row of M_G is normalized using L2-norm and Positive Point-wise Mutual Information transformation is applied to reduce the eventual bias introduced by the conversion towards words with more senses. Finally, *Principal Component Analysis (PCA)* is used to reduce the size of the matrix and achieve WE of size 850. The obtained WNE are evaluated under the task of determining the semantic similarity between pairs of lexical units and obtained results around 15% superior to the ones of Word2Vec with the same evaluation data set SimLex-999 [163]. The main drawback of WNE is that the number of terms in the embeddings model is limited by the size of the semantic graph used to create them. In fact, the model we considered for our retrieval experiments contains only 60K terms: this represents a well-known limit for a NIR system. Hence, we developed an extension of the MatchPyramid architecture, which integrates the missing words with FTE. We call this model MP FTE WNE and its architecture is reported in Figure 4.2. We employ WNE in this model to generate one of the cosine similarity-based matching matrices in Figure 4.2.

In particular, we compute the cosine similarity between each pair of terms in a query and a document, according to this WE model. If one of the terms is missing in the model, we consider the similarity of the pair as 0. Then, we sum this matrix as shown in Figure 4.2, to the one obtained analogously with pre-trained FTE, parametrizing the sum with a scalar coefficient learned by the model.

4.9 Experimental setup

Our experiments are based on the TREC Robust04 collection [140]. The Robust04 corpus is composed of 528,155 documents (news) from TIPSTER Disk 4&5 minus the CR, 249 topics and graded relevance judgments. We perform the retrieval experiments on this collection to reproduce the results obtained with the original MatchPyramid model [35], and also because it is a widely used collection to evaluate NIR systems for ad-hoc retrieval, i.e. DRMM and NVSM. Furthermore, since many NIR models do not scale well either during training or retrieval, it is therefore necessary to evaluate them on small/medium collections. We are aware that, being our evaluation based only on one experimental collection, all our results might not apply to any ad-hoc retrieval collection. However, also we believe that, with our in-depth embedding-based, statistical and topic-by-topic analyses, we are able to overcome the limitations of our experimental setup, and make more general claims. As in the original MatchPyramid paper, we consider only the topic title and the first 500 tokens for each document. We also stemmed all of the terms in the collection using Krovetz Stemmer [5] and removed stopwords using the INQUERY stoplist [129]. The MatchPyramid model used has a total of 4 layers: a convolutional and a pooling layer, followed by a two layer feed-forward neural network. The convolutional kernel has a size of 3×3, while the pooling size is 1×10. The MatchPyramid architecture we adopt to evaluate PWE (MP PWE) and FTE (MP FTE) is shown in Figure 4.1. In MP PWE, we compute the matching matrices using the Wasserstein-Bures dot product described in Section 4.7. The architecture we use for the evaluation of WNE (MP FTE WNE), is depicted in Figure 4.2. Here, as in MP FTE, we employ the standard cosine similarity to compute the matching matrices and keep the same configuration described above for the convolutional, pooling and feed forward layers.

To evaluate retrieval, we consider the P@5, nDCG@5 and MAP measures. This is done to ensure a comprehensive overview of the systems performance, especially in the top part of the rankings which is the most important for the end-user. In particular, P@5 gives us a set-

[5] In order to use pre-trained word embeddings models with stemming, we stemmed all of the terms in the term dictionary of the model and if one stemmed term could be assigned to more than one embedding we computed its embedding as their average

Model	Training corpus	Number of elems.	Emb. size
W2V	Wikipedia	190K	50
FTE	Wikipedia	2.5M	300
WNE	WordNet v.3 [152]	60K	850
PWE	Robust04	76K	1325

Table 4.1 Characteristics of the word embeddings models we employed for our experiments.

based binary measure of the number of relevant documents in the first positions of our runs; nDCG@5 provides a refinement of this measure weighing the order of the documents and graded relevance. The MAP yields an overall evaluation of the rankings, that also takes into account the recall of the systems. We also perform a qualitative evaluation of different WE models: PWE, FTE [6], WNE [7], and Word2Vec Embeddings [8]. The characteristics of these models are reported in Table 4.1. For the PWE training, we consider context windows of size 5 – i.e., we consider the two terms before and after each occurrence of a word as its context. In [36], the PWE were trained on ukWaC, with 2 billion words, and WaCkypedia_EN which is a dump of Wikipedia 2009 [9]. However, we could not train PWE on this large corpora due to the limited computational resources available. Indeed, as we described in Section 4.7, each elliptical probabilistic embedding has 1325 trainable parameters if we consider an embedding space with $d = 50$ dimensions. For this reason, we decided to use the Robust04 collection for the training of the PWE. Despite the limited training data, we were still able to achieve better retrieval results with PWE than with the pre-trained WE on Wikipedia with Word2Vec used in [35]. For reproducibility purposes, we share the code and the WE to perform all our experiments in a public repository. [10]

4.10 Evaluation

WE qualitative evaluation. We conduct a qualitative evaluation of WE. To this end, we select a set of terms related to topic 403: "Osteoporosis", showing their 2D representation according to different WE models. [11] To visualize the relations between elliptical distributions, we adopt the following strategy. We begin performing PCA on the set of mean vectors $\mathbf{m_i}$ associated to the words w_i to visualize. The resulting two-dimensional vectors are the centers

[6] `https://fasttext.cc/docs/en/crawl-vectors.html`
[7] `https://github.com/nlx-group/WordNetEmbeddings`
[8] `https://github.com/pl8787/MatchPyramid-TensorFlow`
[9] Available at: http://wacky.sslmit.unibo.it/doku.php?id=corpora; the pre-trained PWE are not publicly available
[10] `https://github.com/albpurpura/PE4IR`
[11] We use Principal Component Analysis (PCA) as a dimensionality reduction technique to display W2V, WNE, and FTE

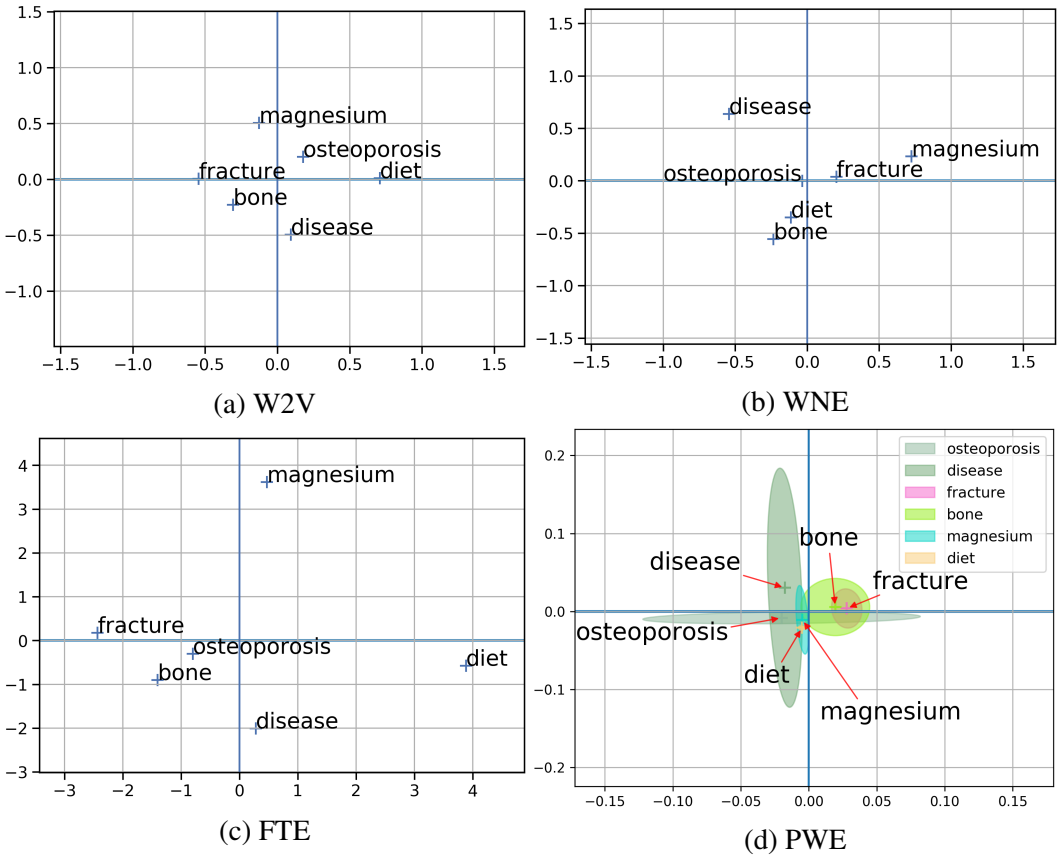


Fig. 4.3 Visualization of a set of terms related to the Robust04 topic 403, according to different WE models.

of the ellipses to be displayed. Then, we consider the precision matrix $\mathbf{P_i}$ [12] associated to each w_i and project its two main eigenvectors to the 2D space identified with PCA. The norms of these projections, multiplied by the normalized corresponding eigenvalues are the sizes of the two semi-axes of the ellipses. Finally, we compute the angle of the ellipses considering the projection of the main eigenvector of each $\mathbf{P_i}$ and the first principal component of PCA. This visualization process preserves – within the limits of the dimensionality reduction operation – the measure of term similarity adopted for the training of the PWE.

The plots in Figure 4.3, show how PWE correctly represents the affinity between the terms selected from the topic description, e.g. between the words “bone”, “disease” and “osteoporosis” – where the ellipses intersections represent the concept that osteoporosis is a bone disease – or “osteoporosis” and “diet”. This however, does not hold for all other models if we consider the cosine similarity between terms – which is proportional to the angle between the words in the scatter plots and is the measure used to compute term similarity

[12] The precision matrix is by definition the inverse of the covariance matrix

Model	P@5	nDCG@5	MAP
MP W2V	0.3711	0.3617	0.1823
MP PWE	$0.4130^{\dagger}$	$0.4060^{\dagger}$	0.1840
MP FTE	$0.4145^{\dagger\dagger}$	$0.3992^{\dagger\dagger}$	0.1952
MP FTE WNE	$\mathbf{0.4161}^{\dagger\dagger}$	$\mathbf{0.4099}^{\dagger\dagger}$	$\mathbf{0.1978}^{\dagger\dagger}$

Table 4.2 Retrieval performance of MatchPyramid with different word embedding models. †† and † indicate a significant difference (paired t-test) from MP W2V with $\alpha = 0.01$ and $\alpha = 0.05$, respectively.

for retrieval. For instance, in Figure 4.3a, we see a correlation in the Word2Vec model between the terms "osteoporosis" and "magnesium" (cosine similarity 0.11) – but not with "disease" (cosine similarity -0.05). Conversely, in the WNE model, the terms "osteoporosis" and "disease" are related (cosine similarity of 0.28) because of their relation in WordNet. However, the WNE model also comes with some drawbacks, since it does not capture the relation between "osteoporosis" and "fracture", most likely because of the longer path that connects the terms in WordNet semantic graph. Finally, the representations of FTE and PWE, are very close in terms of similarity scores between the term "osteoporosis" and the other words considered in the comparison – both models recognize a positive correlation of all the considered terms with "osteoporosis" – but the relative angles with this word differ greatly. In fact, if we only considered the mean vectors of the PWE – represented as the centers of the ellipses – we should have had a completely different set of similarity scores. The final similarity score of PWE is, in fact, obtained thanks to the contribution of the covariance matrices – associated to the probability measure of each term – in the Wasserstein-Bures product. Herein, this similarity component is represented by the overlap between the ellipses. Indeed, the covariance matrix associated to each PWE, allows to represent the correlation between terms in a selective way, along certain directions – associated to their different meanings or usage contexts – without penalizing their representation in the rest of the space, e.g. in the direction of the word senses/contexts which are used less frequently in the training samples.

Retrieval experiments. First, we reproduce the original results obtained on the TREC Robust04 collection by the MatchPyramid model using a pre-trained Word2Vec model [13] (i.e. MP W2V) [35]. We obtain a P@20 of 0.275, very close to the value of 0.272 reported in the MatchPyramid original paper; the MAP and nDCG scores we obtain present an absolute difference with the original ones of less than 0.01. This is likely due to small

[13] `https://github.com/pl8787/MatchPyramid-TensorFlow`

differences in the document parsing and tokenization processes of the collection. Second, we test MatchPyramid employing PWE (MP PWE). The results of our retrieval experiments are reported in Table 4.2. In this case, we observe a 0.04 increase in P@5 and nDCG@5 compared to MP W2V. This improvement is obtained despite the great difference in the number of terms available in Word2Vec and PWE. In fact, there are more than 190K terms in MP W2V, while MP PWE has only 60K terms. Third, we test MatchPyramid employing FTE (MP FTE). In this case there are no OOV terms. We observe a small P@5 increase compared to MP PWE, and a slightly lower nDCG@5 value. Nevertheless, the most relevant difference with MP PWE is in terms of MAP which shows that the words that are missing in PWE have an effect on the number of relevant documents retrieved by the system. This however does not significantly affect the top ranked documents. Finally, we test the extended MatchPyramid architecture (MP FTE WNE) employing two different WE: FTE and WNE. In this case, we get the best results overall. We run a paired t-test between MP W2V (our baseline) and the other described models as reported in Table 4.2. We see that MP FTE WNE is the only system with significant improvements over the baseline for all the considered evaluation measures. MP PWE and MP FTE present significant improvements when we consider only the top ranked documents with P@5 and nDCG@5. Nevertheless, the improvements over the baseline are not as noticeable as we expected. PWE has an impact on NIR architectures, but from a quantitative analysis of the performances, it is not evident where and why PWE provides a gain or a loss.

	P@5	nDCG@5	MAP
MP FTE $\geq$ MP W2V (% topics)	**65%**	**76%**	54%
MP WNE $\geq$ MP W2V (% topics)	**65%**	75%	**57%**
MP PWE $\geq$ MP W2V (% topics)	**65%**	73%	44%

Table 4.3 Proportion of topics (%) where the performance of different variants of the MP model relying on different embedded representations of words is $\geq$ than the performance of MP W2V on the same topic, according to different evaluation measures (P@5, nDCG@5, MAP).

To further investigate this issue, we perform two experiments. First, we report in Table 4.3 the percentage of topics where each variant of the MP model achieves a higher performance – according to different evaluation measures – than the MP W2V model we considered as baseline through this chapter. From this evaluation we observe that almost all variations of the MP model achieve a higher performance than MP W2V in more than 50% of the topics on all performance measures. The only exception is the MP PWE variant when considering the MAP evaluation measure, where we achieve an improvement on 44% of the topics – despite obtaining a higher average MAP value than MP W2V. Overall, this experiment helps us

Topic ID	Title	Topic ID	Title
312	Hydroponics	447	Stirling engine
328	Pope beatifications	630	Gulf war syndrome
348	Agoraphobia	647	Windmill electricity
353	Antartica exploration	657	School prayer banned
403	Osteoporosis	659	Cruise health safety
444	Supercritical fluids	697	Air traffic controller

Table 4.4 Topics considered for the topic-level qualitative evaluation.

assess how the performance improvements we observed earlier on the averaged performance measures translate to the topic level and on the overall user satisfaction.

Second, we perform a more detailed topic-by-topic evaluation of the considered systems. In Figure 4.4, we compare, for each topic, the AP of MP W2V against MP PWE and MP FTE WNE. In the figure, if a point lies below the red bisector, it means that the system associated to the x-axis has a higher AP than the one on the y-axis (and viceversa). Whereas, the nDCG@5 performance difference between MP W2V and MP PWE or MP FTE WNE is reported in Figures 4.5b and 4.5a, respectively. The stem plots indicate the nDCG@5 score difference between the systems appearing on the y-axis. If the value is positive (green), the system associated to the positive side of the axis performs better than the one associated to the negative one with reference to the topics indicated on the x-axis (and vice versa for red points). Relying on the data visualized in Figures 4.4 and 4.5, we selected twelve topics – reported in Table 4.4 – where MP W2V and MP PWE show a sizeable performance difference both in terms of AP and nDCG@5.

First of all, in Figure 4.4 we notice that using different WE affects the performance on the selected topics. For instance, we observe a considerable performance difference in Figure 4.4a on topic 447 where MP W2V has an AP of 0.0, while MP FTE has an AP close to 0.65. A similar situation is true for topics 328 and 444 in Figure 4.4b, where the performance of MP PWE is over 0.60 and the AP of MP W2V is below 0.05. If we consider topics 403 and 630 in Figures 4.4a and 4.4c, we also notice the improvement obtained by combining FTE and WNE compared to relying only on FTE. In fact, the conjunct use of FTE and WNE improves the AP of about 0.10 with respect to employing FTE alone. Conversely, in other topics such as 447 or 444 (see Figures 4.4a and 4.4c) we observe a performance deterioration because MP FTE WNE cannot rely on WordNet for additional information since the terms “supercritical” and “stirling” are not contained in WNE. If we examine the stem plots in Figure 4.5a and consider the nDCG@5 difference for each topic of MP W2V and MP FTE WNE, we notice that there are only three topics where the nDCG@5 of MP W2V is evidently higher (≥ 0.5) than the nDCG@5 of MP FTE WNE. These topics are 353, 647, and 657. In these cases, the lower performance of MP FTE WNE is likely due

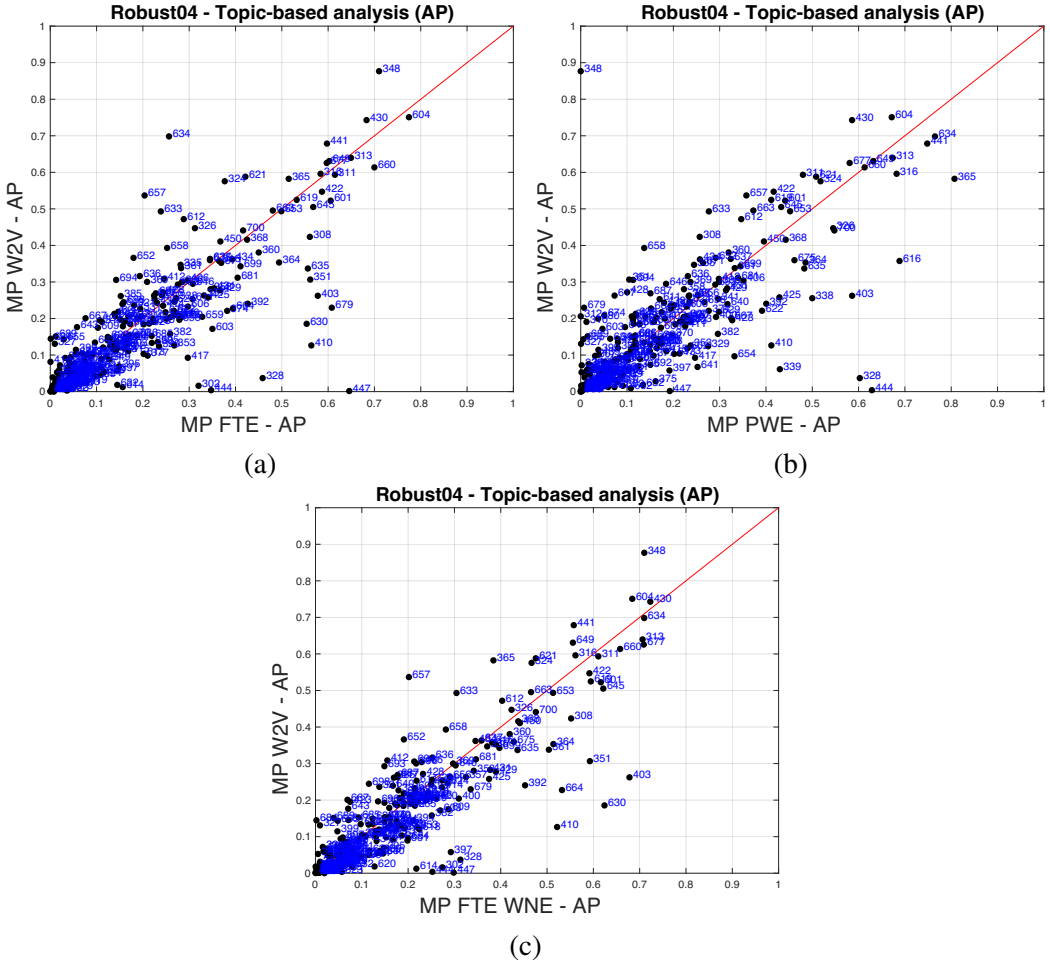
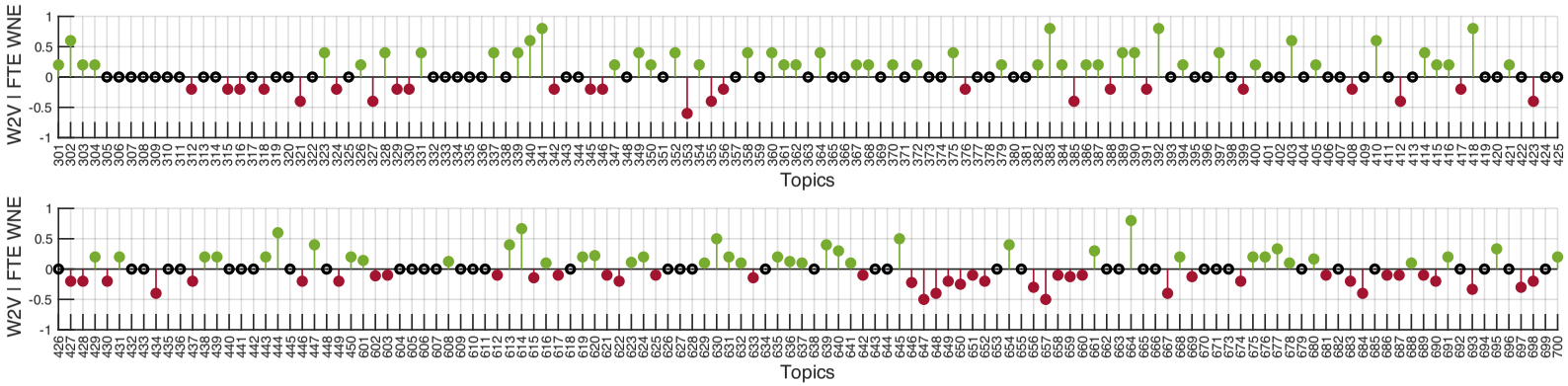


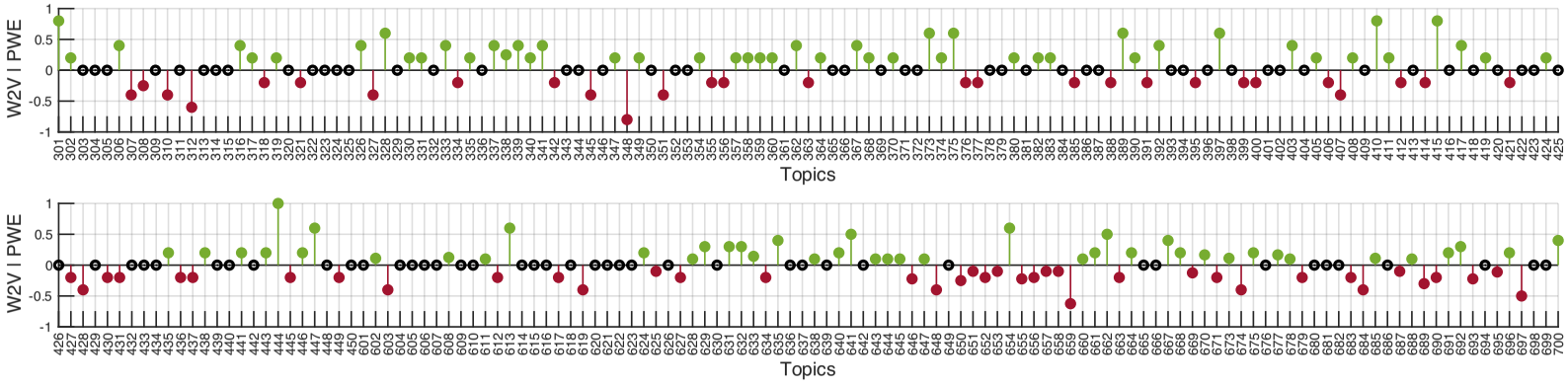
Fig. 4.4 Comparison of the AP of different MatchPyramid models employing FastText Embeddings (MP FTE), Probabilistic Word Embeddings (MP PWE) and FastText + WordNet Embeddings (MP FTE+WNE), against MatchPyramid with Word2Vec embeddings (MP W2V).

to the lack of the query terms “antartica”, “windmill” and “banned” in WNE. When we compare MP W2V with MP PWE in Figure 4.5b, the topics where the former system’s nDCG@5 is over 0.5 higher than the one of the latter are: 312, 348, 659, and 697. The lower performance of MP PWE associated to topics 312 and 348, is due to the fact that all of the query terms are missing from the WE model. On the other hand, the performance difference on the topics 659 and 697, is likely due to the small training set of PWE. In fact, the key terms “cruise” and “controller” of topics 659 and 697 only appear respectively 3K and 1.5K times in the training corpus – while the average frequency of the other terms in these topics is 53K. Hence, the lack of training contexts for these terms, which are the most important in both of the topics, led to a poor training of the corresponding WE and

consequently to a low retrieval performance. The experiments performed in this section, highlight the potential of PWE in NIR systems. In fact, the qualitative evaluation of PWE in Figure 4.3d, underscores the role of the covariance matrix in the WE model. This component allows to balance the contribution of the mean vector in the similarity function we employed – the Wasserstein-Bures normalized dot product – adjusting it for the different contexts where a word is used in the training corpus. The potential of PWE emerged also when we employ this WE model for retrieval, where we obtained a statistically significant improvement in the top part of the rankings, compared to Word2Vec embeddings. Moreover, through the topic level analysis we performed, we could analyze and motivate why PWE led or did not lead to a performance improvement w.r.t. the baseline system employing Word2Vec embeddings. The potential of this innovative WE is also confirmed by the positive results obtained combining FTE with WNE which have similar properties to PWE. Indeed, the addition of WNE leads to the best performances overall, which is only constrained by the number of terms in the ontology used to train them.



(a) MP W2V and MP FTE WNE nDCG@5 difference by topic.



(b) MP PWE and MP W2V nDCG@5 difference by topic.

Fig. 4.5 The stem plots indicate the nDCG@5 score difference between the systems indicated on the y-axis. If the value is positive (green), the system associated to the positive side of the axis (MP FTE WNE in 4.5a and MP PWE in 4.5b) performs better than the one associated to the negative one (and vice versa, red points) on the topic indicated on the x-axis.

4.11 Final Remarks

In this chapter, we analyzed the performance in ad-hoc retrieval, of Elliptical PWE. This type of WE is able to encode richer relations among terms [36] and to improve retrieval performance by generating meaningful similarity matching signals between query and document terms. To evaluate the generalization power of PWE, we first performed a qualitative evaluation of the WE, visualizing the elliptical probability distributions. Then, we employed a NIR architecture, MatchPyramid [35], to evaluate the quality of the matching signals, generated with different WE models, for the retrieval task. We selected this architecture for its capacity to compute the relevance score for a query-document pair, considering only the similarity scores between each term from the two texts, without any alteration. In this way, we eliminated any potential bias against different WE models with distinct characteristics, e.g. size, sparsity, or structure. We also extended the MatchPyramid model (MP FTE WNE) to employ two types of WE with different characteristics at the same time: fastText [150] and WordNet embeddings [37]. The former embedding model is based on character n-grams embeddings, which allowed us to obtain a semantically meaningful representation of all the terms in our experimental collection, eliminating the OOV problem. The latter, allowed us to enrich the matching signals with broader conceptual-semantic and lexical relations form WordNet.

Although our results employing PWE do not yet lead to a sizeable improvement over traditional Word2Vec embeddings, we believe PWE can nonetheless be considered a promising word vector representation of terms for NIR. In fact, analyzing the similarity matrices associated to these embeddings and the document rankings topic-by-topic, we observed that PWE can be a competitive word representation for retrieval, outperforming in many cases a pre-trained Word2Vec model. This intuition is confirmed by the results obtained with the extended MatchPyramid model that leverages on a word representation which encodes richer term relations to improve retrieval – i.e. MP FTE WNE. MP FTE WNE achieved, whenever the query terms were present in the WordNet embeddings models, a better performance than Word2Vec or FTE. This suggests that investing in WE models – such as WordNet embeddings or PWE – able to better and efficiently represent hypernymy relations could help improving retrieval performance. On the other hand, the large number of parameters of PWE, makes them difficult to use and train, especially on small collections. In fact, we encountered numerous engineering challenges to optimize retrieval and training time for this type of WE, and we could not train or use them on a larger collection.

Chapter 5

Probabilistic Training Strategies

5.1 Introduction

In this chapter and in the remainder of this thesis we will shift our focus towards LETOR models. In fact, despite the ability of NIR models to perform semantic matching between query and document terms, these approaches suffer from significant efficiency problems which render their adoption in real-world search systems quite expensive and therefore less competitive than lexical models.

LETOR models on the other hand, allow to efficiently compare query-document pairs relying on inexpensive sets of features which can be computed in a distributed and efficient way. These sets of features can also come from different sources and therefore represent more useful aspects for determining the relevance of an item. LETOR systems can also be combined to NIR models when necessary, adding a minimal computational overhead. For this reason, LETOR models have been largely employed in the industry in the past few years, reaching their popularity peak in the early 2010s. Nowadays, they are seeing a renewed interest thanks to the recent advancements in the deep learning domain. Indeed, if early LETOR models most frequently relied on *Support Vector Machine (SVM)*s or *Random Forest (RF)* models, now they employ more sophisticated and effective deep learning architectures. This technological shift was encouraged by the increased availability of training data, the development of new training strategies and the recent availability of more powerful hardware to support these models.

In this chapter we present and evaluate a new family of training strategies for neural ranking models. We propose a new way to represent the relevance of a query-document pair, relying on relevance judgments distributions. Despite being easily applicable to any ranking model, we evaluate here the impact of the proposed strategies on LETOR systems.

Indeed, the development and evaluation of new training strategies for ranking models has been the focus of many research works in this domain for the past few years. For this reason, we believe this is the best evaluation scenario to show the advantages offered by our training strategy.

5.2 Research Question

Given an unordered set of items, the ranking problem consists in ordering them according to a certain notion of relevance. Generally, in IR we rely on a notion of relevance that depends on the information need of a user, expressed through a keyword query. Given a document corpus and a set of topics, the corresponding relevance judgments are obtained by asking different judges to assign a relevance score to each document-topic pair. Multiple judges usually assess the same document-topic pair, and the final relevance label for the pair is obtained by aggregating these scores [164]. This process is a cornerstone for system training and evaluation and has contributed to the continuous development of IR, especially in the context of international evaluation campaigns. Nonetheless, the opinion of different judges on the same document-topic pair might be very different or even diverge to the opposite ends of the spectrum – either because of random human errors or due to a different interpretation of a topic. Inevitably, the aggregation process conflates the multiple assessors viewpoints on document-topic pairs onto a single one, thus losing some expressive power – even though it also reduces annotation errors and outliers. Our research hypothesis is that *Machine Learning (ML)* models – i.e., LETOR [165] and NIR [12] models – could use all the relevance labels collected in the assessment process to improve the quality of their rankings. Indeed, judges disagreement on a certain document-topic pair can be due to an inherent difficulty of the topic or to the existence of multiple interpretations of it. For this reason, we argue that designing ML models able to learn from the whole distributions of relevance judgments could improve the models' generalization power and their overall performance through the usage of this additional information.

Our hypothesis can be summarized by the following research question.

> **RQ3:** Can we improve the performance of LETOR models by leveraging the entire information provided by relevance judgments distributions?

5.3 Methods

We interpret the output of a LETOR model as a probability value or a probability distribution – according to the experimental hypotheses – and define different *Kullback–Leibler (KL)* divergence-based loss functions to train the model using the distribution of relevance judgments obtained for the current training instance. Such training strategy allows us to leverage all the available information from the human judges without additional computational costs compared to traditional LETOR training paradigms.

The proposed loss functions could be used to train any ranking model that relies on gradient-based learning, including popular NIR models such as DRMM, MatchPyramid or LETOR ones such as the recently proposed *Data Augmented Self-Attentive Latent Cross ranking network (DASALC)* [166]. However, to reduce the complexity of the factors coming into play in our evaluation, we focus on one neural LETOR model and one decision tree-based *Gradient Boosting Machine (GBM)* model – the model at the base of the popular LambdaMART [22] ranker and used as a strong baseline in many recent LETOR research papers such as [21, 20, 167, 19, 166]. We assess the quality of the proposed training strategies on four standard LETOR collections – i.e. MQ2007, MQ2008, MSLR-WEB30K [78] and OHSUMED [79] described in Section 2.8. To further validate our hypotheses, we also conduct a crowdsourcing experiment to build a new LETOR collection based on the COVID-19 MultiLingual Information Access (MLIA) data. [1] We then use the raw relevance labels and their aggregated version to assess the impact on a LETOR model trained on raw relevance labels versus their aggregated form.

5.4 Contributions

Our contributions towards the aforementioned research question are the following.

- we define five new loss functions to train different LETOR models using probability distributions of relevance labels;
- we evaluate the proposed loss functions on different publicly available LETOR collections using different LETOR models;
- we also evaluate the proposed loss functions on a newly created crowdsourced LETOR test collection based on COVID-19 Multi-Language Information Access (MLIA) data that we publicly share.

[1] `http://eval.covid19-mlia.eu`

5.5 Outline

This chapter is organized as follows. In Section 5.6, we describe the most relevant training strategies for ML ranking models; in Section 5.7, we present the probabilistic loss functions leveraging relevance judgments distributions that we propose and the neural LETOR model we employ for their evaluation; in Section 5.8, we describe our experimental setup and in Section 5.9 we discuss the evaluation results; in Section 5.10, we present our final remarks on the results we presented in this chapter.

5.6 Related Work

Since the early 2010s, decision tree-based approaches such as LambdaMART [22] have been the most popular ML models in the LETOR domain, but recently – thanks to the growing size of LETOR collections, new optimization functions [19] and feature normalization strategies [33] – deep learning approaches, transformer-based ones in particular, are showing an increasingly competitive performance compared to previous strategies [166].

Transformer-based LETOR models rely on one or more self-attention layers [29]. This type of neural layer architecture was originally proposed in [29], and later popularized by language models such as BERT [27]. This architecture allows LETOR models to efficiently evaluate and compare lists of candidate relevant documents to a user query, providing a numerical estimate of their relevance. One of the latest and most successful approaches of this kind is DASALC [166]. It relies on a few strategies such as neural feature transformation, self-attention layers, a listwise ranking loss and model ensembling, to outperform strong non-neural baselines such as LambdaMART on public LETOR collections. Strategies such as neural feature transformation are frequently employed in the context of neural LETOR models to normalize the representation of their inputs so that they could be better interpreted by such models [33]. Self-attention layers allow to efficiently compare groups of items at the feature level. Finally, listwise ranking losses and model ensembling strategies are already popular solutions in LETOR and machine learning to improve the performance of ranking models [22, 21].

The group of Transformer-based neural LETOR models we consider in this chapter is part of a new wave of approaches compared to the strategies which were popular in the early 2010s. If previously the focus was mostly on the development of an optimization framework able to approximate IR metrics which are – in their standard definition – not differentiable, now the focus shifted towards the development of novel neural architectures relying on simpler loss functions and more standard training strategies for gradient descent-

based training. This strategy allows to simplify the overall training process of a ranking model, while at the same time achieving a better performance relying on the larger availability of training data. For this reason, Transformer-based models are overall more similar to each other than previous approaches for what concerns the type of architecture they employ, while they differ mostly in the strategies used to manage the input data and in the loss function formulation.

In this chapter, we will show how we can improve the performance of LETOR models in an efficient way through the usage of a new class of *probabilistic* loss functions that we propose. Indeed, an important line of research in the LETOR domain traditionally focused on the development of new loss functions to train ML models [165]. These loss functions are generally categorized as *pointwise*, *pairwise* and *listwise* [168]. Pointwise loss functions are used to train a model to fit the corresponding relevance score for each document-topic pair as in a regression task. Loss functions belonging to this class can be described using the following general formulation:

$$\text{Pointwise}(q,d,y) = f(s(q,d),y) \tag{5.1}$$

where q indicates a query, d a document, y its relevance label, $s(\cdot,\cdot)$ the function learned by a ML model to compute the relevance of a document given a query, and $f(\cdot)$ the generic function which compares the score computed by the model that is being trained with the corresponding relevance label. One of the possible implementations of $f(\cdot)$ is the MSE [169]. Pairwise loss functions consider pairs of documents and compare their relevance scores using different strategies. This class of losses can be formalized as:

$$\text{Pairwise}(q,d_1,d_2) = f(s(q,d_1),s(q,d_2)), \tag{5.2}$$

where the function f can have different formulations such as the Hinge function $\phi(z_1,z_2) = max(0,1-z_1+z_2)$, where $z_i = s(q,d_i)\ \forall i \in \{1,...,n\}$ [15]. Finally, listwise loss functions take into account a set of documents relative to a certain query and compute the loss for the group of items as:

$$\text{Listwise}(q,(d_1,...,d_n),(y_1,...,y_n)) = f((s(q,d_1),...,s(q,d_n)),(y_1,...,y_n)), \tag{5.3}$$

where $(y_1,...,y_n)$ are the relevance judgments associated to $(d_1,...,d_n)$ and the function $f(\cdot)$ can take different formulations such as the ApproxNDCG [170] loss.

Amongst the numerous loss function formulations proposed by the LETOR community, the most widely-employed in the latest state-of-the-art text-based NIR [28] and LETOR models [33] are the pairwise Hinge [12] and the listwise ApproxNDCG loss [170, 21], respec-

tively. The formulation of the Hinge loss as used in the NIR domain is $\text{Hinge}(q,d^+,d^-) = \max(0,1-s(q,d^+)+s(q,d^-))$, where d^+ and d^- identify respectively a relevant and a not-relevant document for the query q. The goal of this loss function is to maximize the difference between the relevance probabilities – indicated by the function $s(q,\cdot)$ – computed for each document. The Hinge loss is frequently chosen to train text-based NIR models such as DRMM [15], MatchPyramid [35] or the most recently proposed CEDR [28], but was also used as a baseline in LETOR research works such as in [171], where it demonstrated to be a competitive candidate among other pointwise or pairwise loss functions. The main advantage of this loss function is its ability to perform well on relatively small datasets – as it is often the case for text-based NIR models when evaluated on shared IR test collections. Differently from the Hinge loss, the ApproxNDCG loss can take into account more than two documents at a time and, as the name suggests, provides a differentiable approximation of the nDCG measure for the evaluation of a ranked list. Given a permutation π of items $\{x_1,...,x_n\}$ and the corresponding sequence of relevance labels $\mathbf{y}$, the nDCG is defined as:

$$\text{nDCG}(\pi,\mathbf{y}) = \frac{\text{DCG}(\pi,\mathbf{y})}{\text{DCG}(\pi^*,\mathbf{y})}, \tag{5.4}$$

where π^* is the ideal ranking of the items according to the relevance labels $\mathbf{y}$. The DCG is computed as [168]:

$$\text{DCG}(\pi,\mathbf{y}) = \sum_{i=1}^{n} \frac{2^{y_i}-1}{\log_2(1+\text{r}(x_i,\pi))}, \tag{5.5}$$

where $\text{r}(x_i,\pi)$ is the function returning the rank of item x_i in π and y_i is its label. This measure, however, is not differentiable because the function $\text{r}(\cdot,\cdot)$ is not. To tackle this problem, Qin et al. [170] propose a differentiable approximation of nDCG which can be used as a loss function to train ML models [21]. In the ApproxNDCG loss, the non-differentiable $\text{r}(\cdot,\cdot)$ function is replaced by its approximation:

$$\hat{r}(x_i,\pi) = 1 + \sum_{j\neq i} \mathbf{I}_{s(x_i)<s(x_j)}, \forall x_j \in \pi, \tag{5.6}$$

where $s(\cdot)$ indicates the scoring function of the model we are training and $\mathbf{I}_{u<v}$ is the indicator function which is 1 if $u<v$ and 0 otherwise. In turn, the indicator function is approximated with: $\sigma(v-u) = (1+e^{-\alpha(v-u)})^{-1}$, where α is a parameter to control the steepness of the sigmoid function $\sigma(\cdot)$.

More recently, a stochastically treated version of the ApproxNDCG function was proposed by Bruch et al. [19]. In the paper, zero-mean logistic noise is added to the sigmoid function presented earlier, which becomes $\sigma((v-u)+Z_{vu})$, where $Z_{vu} \sim \text{Logistic}(\text{mean} = 0, \text{scale} = \beta)$. Intuitively, adding random noise to the model outputs induces it to increase the difference between the relevance scores of relevant and not-relevant items to maintain their relative ordering in the final ranked list.

ApproxNDCG and its variant with *Stochastic Treatment (ST)* are the most popular and best performing listwise loss functions among the recently published neural LETOR works such as [21, 19, 172, 33]. According to our evaluation on several test collections, they also outperform RankNet, ListMLE [8] and ApproxMRR losses.

Regarding the collection of relevance judgments, there is still an open debate in the IR community on how to reliably collect them and on the best strategies for their aggregation. Crowdsourcing is a valuable option in this context [173]. The TREC Crowdsourcing track [174–176] for example, explored the challenges related to the collection and management of relevance judgments for Web pages and search topics. Numerous aggregation options are also described in the IR literature on crowdsourcing [31, 164, 32] involving strategies to weigh the annotations of each judge depending on the topic difficulty and/or his/her level of confidence on it. Here, we take a new approach to this problem, eliminating the relevance judgments aggregation step and training a model on the distribution of relevance judgments associated to each document-topic pair.

We propose five different loss functions that allow ranking models to take advantage of relevance judgments distributions prior to their aggregation. Since, to the best of our knowledge, no similar method was previously presented in the literature, we compare the newly proposed training strategies to the four most representative and best performing loss functions described above (MSE, Hinge, ApproxNDCG and ApproxNDCG with ST).

Given the recent success of self-attention-based neural models in the LETOR domain [19, 21, 33, 34, 166], we evaluate the aforementioned loss functions on a similar neural LETOR model serving as our main reference architecture. Despite this, decision tree-based models remain a strong baseline across different collections [166]. For this reason, we extend the evaluation of the proposed loss functions also to a decision tree-based GBM model, extending the LightGBM [2] framework to employ the proposed loss functions.

5.7 Proposed Approach

In this section, we introduce the proposed probabilistic loss functions [3] and the neural LETOR model that we use as reference architecture.

Pointwise Loss Functions. Firstly, we present a pointwise loss function relying on KL divergence to compare two probability distributions, i.e. the relevance score computed by our LETOR model for a document-topic pair and its corresponding true relevance label. In

[2] `https://github.com/microsoft/LightGBM`

[3] We define them as "probabilistic" because they rely on distributions of relevance judgments and not on single labels

this case, we interpret the relevance label assigned to a document as if it was generated by a Binomial random variable modeling the judges' annotation process. We assume that n assessors provided one binary relevance label for each document-topic pair, i.e., to state whether the pair was a relevant or a not-relevant one. This process can be modeled as a Binomial random variable $P \sim \text{Bin}(n, p)$ where the success probability p for each sample is the average of the binary responses provided by the judges. Since in most LETOR datasets, the relevance judgments are not real values in the $[0, 1]$ range, we normalize them to fit this interval. In other words, if, for example, a dataset contained integer relevance labels such as $\{0, 1, 2\}$, the normalized relevance labels would be $\{0.0, 0.5, 1.0\}$. This normalization of the relevance labels is only used during training and does not impact the documents' relative ordering in a ranked list.

We then apply the same reasoning for the interpretation of our model output probability score as another Binomial distribution $\hat{P} \sim \text{Bin}(n, \hat{p})$ with the same parameter n – empirically tuned for the numerical stability of the gradients during training – and probability $\hat{p}$ equal to the output of the model which is kept in the $[0, 1] \in \mathbb{R}$ range employing a sigmoid activation function at the output layer. At this point, the model output and relevance labels distributions are expressed as Binomial distributions with the same parameter n and different success probabilities; hence, we can compare them using the KL divergence:

$$
\begin{aligned}
&D_{KL_{Bin}}(P||Q) = \sum_{x \in \mathscr{D}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \\
&= \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \log \left(\frac{p^k (1-p)^{n-k}}{\hat{p}^k (1-\hat{p})^{n-k}} \right) \\
&= \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} \left(k \log \left(\frac{p}{\hat{p}} \right) + (n-k) \log \left(\frac{1-p}{1-\hat{p}} \right) \right) \\
&= \sum_{k=0}^{n} \left[\binom{n}{k} p^k (1-p)^{n-k} k \log \frac{p}{\hat{p}} \right] + \sum_{k=0}^{n} \left[\binom{n}{k} p^k (1-p)^{n-k} (n-k) \log \frac{1-p}{1-\hat{p}} \right] \\
&= \left(\log \frac{p}{\hat{p}} \right) \sum_{k=0}^{n} \left[k \binom{n}{k} p^k (1-p)^{n-k} \right] + \left(\log \frac{1-p}{1-\hat{p}} \right) \sum_{k=0}^{n} \left[(n-k) \binom{n}{k} p^k (1-p)^{n-k} \right] \\
&= \log \left(\frac{p}{\hat{p}} \right) np + \log \left(\frac{1-p}{1-\hat{p}} \right) n(1-p),
\end{aligned}
\tag{5.7}
$$

where the last equation follows from the definition of expected value of Binomial distributions with parameters p and $(1-p)$ and is differentiable because it is the sum of two differentiable and continuous functions for p and $\hat{p}$ in the open $(0, 1)$ interval. Since KL divergence is not symmetric – which would lead to issues in the comparison of different items during model optimization – we employ the following symmetric and non-negative formulation as the loss

function for each data point in a training batch:

$$\text{Pointwise}_{KL(Bin)} = (D_{KL}(P_i||Q_i) + D_{KL}(Q_i||P_i)) * w_i. \tag{5.8}$$

Since we train our model feeding it all the items corresponding to one or more ranked lists provided in the training data, we need to balance the contributions of relevant (the minority) and not-relevant (the majority) items to the final loss function value. For this reason, before computing the total loss value in a batch by summing the contribution of each data point, we rescale each term by a factor w_i, inversely proportional to the number of times an item of the same class appeared in the batch.

For instance, if a batch contains 2 relevant items and 3 non-relevant ones, we will rescale the $\text{Pointwise}_{KL(Bin)}$ value associated to the relevant samples by a factor of 0.5 while the contributions of not-relevant items will be rescaled by a factor of 0.3. Since our relevance labels are graded and normalized between $[0,1] \in \mathbb{R}$, we consider as not-relevant the data points associated to a relevance label lower than an empirically set threshold, and the remaining ones as relevant. In our experiments, we set this threshold to 0.1 – instead of for example 0.0 – for it to be used also when simulating the relevance labels distributions as described in Section 5.9, sampling them from continuous probability distributions.

Whenever an actual relevance judgments distribution is available, we propose to use another pointwise loss function which takes into account the distribution of values over a number of relevance grades, interpreting them as outcomes from a Multinomial distribution [177, 178] $P \sim \text{Mul}(n, (p_1, ..., p_k))$ – which is a generalization of the aforementioned Binomial distribution with the same parameter n. The outcomes of the modeled random process take a finite number of values, where $\sum_{i=1}^{k} p_i = 1$. Each p_i indicates the probability of one of the k possible relevance grades to be selected by the pool of judges that were employed to asses a certain document-topic pair. To obtain a comparable distribution to the output of our model we adjust the output layer size to k and employ a softmax activation function over each output sequence obtaining the probabilities $(\hat{p}_1, ..., \hat{p}_k)$, which allow us to define the random variable $\hat{P} \sim \text{Mul}(n, (\hat{p}_1, ..., \hat{p}_k))$. We then compare the two distributions with the same strategy used before, changing the formulation of the KL divergence to Multinomial distributions:

$$D_{KL(Mul)}(P||\hat{P}) = \sum_{i=j}^{k} p_j \log \frac{p_j}{\hat{p}_j}. \tag{5.9}$$

This function is continuous and differentiable for non-zero probability values but, again, not symmetric. Therefore, we train our model using the following symmetric and non-negative

formulation to evaluate the quality of approximation of the relevance of each item:

$$\text{Pointwise}_{KL(Mul)} = \left(D_{KL(Mul)}(P_i||\hat{P}_i) + D_{KL(Mul)}(\hat{P}_i||P_i)\right) * w_i. \tag{5.10}$$

As in the previous case, before computing the total loss in a batch, we rescale the contribution of each data point in it by a factor w_i.

In this context, we recommend the collection of a number of relevance labels for each query-document pair sufficient to estimate the distribution of opinions on its relevance for the user base of the search system. For example, if we are evaluating the relevance of a recently published academic study with respect to a certain topic, if our audience is made of experts from the same academic field, then a wider range of relevance labels should be collected to capture all of the nuances of the opinions field professionals could have on the topic. On the other hand, if our audience is the general public, the number of relevance judgements required can be smaller and proportional to the public agreement on the specific topic.
Note also that, despite the interpretation we provide for the parameter n in the Binomial and Multinomial distributions as the number of judges providing relevance labels for the same query-document pair, in the remainder of this paper we consider n as a hyperparameter of the model. Indeed, even if there could be a relation between the number of judges providing relevance labels and the corresponding random variable which could model this process, this intuition would be hard to verify empirically. In fact, the number of judges available for the creation of a new IR collection is often limited by real-world constraints – such as labor cost or the availability of an appropriate number of trained professionals – therefore, our hypothesis becomes hard to verify for a large number of topics where potentially dozens or even hundreds of judges could be required for the annotation of each query-document pair in an exhaustive study.

Pairwise Loss Functions. We also propose two new pairwise loss functions. The principle underlying these loss functions is to encourage a LETOR model to increase the distance between the distributions associated to a relevant and a not-relevant document, while allowing some slack between them. The idea is to increase the model robustness and generalization power – similarly to the stochastically treated ApproxNDCG loss [19] – through a comparison between distributions instead of real valued relevance scores. To achieve this goal, given a topic, we compare all pairs of relevant and not-relevant documents in a batch by considering the output relevance scores produced by our model.

In other words, for each topic in an experimental collection we consider every pair of documents available with a different relevance label – obtained by either aggregating relevance scores if relevance judgments distributions are available, or taking their exact value – and compare them to train a LETOR model with one of the loss functions described below.

We evaluate two possible interpretations for these relevance scores. The first option, similarly to what we did for the Pointwise$_{KL(Bin)}$ loss function, is to consider the relevance scores as the success probabilities of Binomial random variables and then to compute their KL divergence.

The second option is to consider the relevance scores of each pair of data points as samples from two different Gaussian random variables $P \sim \mathcal{N}(\mu_p, \sigma)$ and $Q \sim \mathcal{N}(\mu_q, \sigma)$ with the same standard deviation σ but centered on the model output scores μ_p and μ_q. Our hypothesis is that relevance judgments provided by different judges for a certain document-topic pair (relevant or not-relevant) will have a certain standard deviation (σ) and will be all centered around a certain value μ, following a Gaussian random process. For example, given the set of possible relevance labels $\{0,1,2\}$, we expect that a relevant document-topic pair will be, with a relatively high probability, associated to the labels 1 or 2, while a not-relevant one will have a lower average relevance label associated to it. Therefore, if we assume the standard deviation – i.e. the disagreement of different judges over each annotated sample – to be constant over time, we can model the process with a Gaussian random variable with μ equal to the output relevance score of our model – interpreted as a sample from the true distribution – and standard deviation σ to be adjusted according to the level of agreement/disagreement that we hypothesize in the annotation process. The unbounded range of values that a Gaussian random variable can take is not a concern to us since we are using it to compare the relative ordering of items and not the absolute values of the scores returned by our model.

Depending on the modeling strategy, the proposed loss functions take the following formulations typical of pairwise hinge losses [168], where we replace the term dedicated to compare a pair of item with their sign-corrected KL divergence value:

$$\text{Pairwise}_{KL(Bin)} = \max(0, m - \text{sign}(p^+ - p^-) D_{KL(Bin)}(P^+_{Bin}, P^-_{Bin}), \tag{5.11}$$

where m is a slack parameter to adjust the distance between the two distributions, and $P^+_{Bin} \sim \text{Bin}(n, p^+)$ and $P^-_{Bin} \sim \text{Bin}(n, p^-)$ are two Binomial distributions corresponding to a relevant and to a not-relevant document-topic pair, respectively. The respective success probabilities are equal to the sigmoid-bounded relevance probability scores returned by the LETOR model to train, as done for the corresponding pointwise loss function, see eq. (5.8). In this case, if the relevant data point has a relevance probability $p^+ > p^-$, the loss is equal to the difference between m and the KL divergence between the distributions. This difference is lower bounded by zero, thanks to the max operation implemented with the *Rectified Linear Unit (ReLU)* function. In this situation, the slack variable m can be tuned to increase or decrease the distance between the two distributions. Conversely, if $p^- > p^+$, then the loss

is positive and equal to the sum of m and the value of the KL divergence between the two distributions.

The formulation of the pairwise loss function relying on Gaussian distributions has a similar form, but it uses the formulation of the KL divergence between to Gaussian random variables $P \sim \mathcal{N}(\mu_p, \sigma_p)$ and $Q \sim \mathcal{N}(\mu_q, \sigma_q)$:

$$
\begin{aligned}
D_{KL(\mathcal{N})}(P||Q) &= \int P(x)\log\left(\frac{P(x)}{Q(x)}\right)dx = \int P(x)\log\left(\frac{\frac{1}{\sigma_p\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu_p}{\sigma_p})^2}}{\frac{1}{\sigma_q\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu_q}{\sigma_q})^2}}\right)dx \\
&= \int P(x)\log\left(\frac{\sigma_q}{\sigma_p}\right)dx + \int P(x)\left[-\frac{1}{2}\left(\frac{x-\mu_p}{\sigma_p}\right)^2 + \frac{1}{2}\left(\frac{x-\mu_q}{\sigma_q}\right)^2\right]dx \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\int P(x)(x-\mu_p)^2dx + \frac{1}{2\sigma_q^2}\int P(x)(x-\mu_q)^2dx \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2} + \frac{1}{2\sigma_q^2}\int P(X)(x-\mu_p+\mu_p-\mu_q)^2dx \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2} + \frac{1}{2\sigma_q^2}\left(\int P(X)(x-\mu_p)^2dx\right. \\
&\left.+(\mu_p-\mu_q)^2\int P(x)dx + 2(\mu_p-\mu_q)\int (x-\mu_p)P(x)dx\right) \\
&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2} + \frac{1}{2\sigma_q^2}\left(\sigma_p^2 + (\mu_p-\mu_q)^2\right) \\
&= \frac{1}{2}\left[2\log\frac{\sigma_q}{\sigma_p} - 1 + \frac{\sigma_p^2+(\mu_p-\mu_q)^2}{\sigma_q^2}\right].
\end{aligned}
\tag{5.12}
$$

This function is differentiable and, if the two random variables have the same variance, also symmetric and not-negative. Therefore, we can employ it to train the model:

$$
\text{Pairwise}_{KL(\mathcal{N})} = \max(0, m - \text{sign}(p^+ - p^-)D_{KL(\mathcal{N})}(P^+_{\mathcal{N}}, P^-_{\mathcal{N}})). \tag{5.13}
$$

Listwise Loss Function. Finally, we propose a listwise loss function also based on the KL divergence between distributions. In this case, we consider the whole set of relevance probabilities associated to k documents in a ranked lists of a batch and their respective relevance labels as two multivariate Gaussian distributions. In a similar way as in eq. (5.12), we can compute the KL divergence between two multivariate Gaussian distributions $P \sim \mathcal{N}(\boldsymbol{\mu_p}, \Sigma_p)$ and $Q \sim \mathcal{N}(\boldsymbol{\mu_q}, \Sigma_q)$, obtaining:

$$
D_{KL(\mathcal{N}_{mult})} = \frac{1}{2}\left[\text{tr}\left(\Sigma_q^{-1}\Sigma_p\right) + (\boldsymbol{\mu_q}-\boldsymbol{\mu_p})^T\Sigma_q^{-1}(\boldsymbol{\mu_q}-\boldsymbol{\mu_p}) - k + \log\left(\frac{\det\Sigma_q}{\det\Sigma_p}\right)\right]. \tag{5.14}
$$

If the two distributions have the same diagonal covariance matrix, eq. (5.14) reduces to:

$$D_{KL(\mathcal{N}_{mult})} = \frac{1}{2}(\boldsymbol{\mu_q} - \boldsymbol{\mu_p})^T \Sigma_q^{-1} \left[(\boldsymbol{\mu_q} - \boldsymbol{\mu_p})\right], \tag{5.15}$$

which is also symmetric and not-negative. As for the Pointwise$_{KL(Bin)}$ and Pointwise$_{KL(Mul)}$, we rescale the components of the distributions associated to relevant and not-relevant items by a factor $\boldsymbol{w}$ inversely proportional to their respective class frequency in the current ranked list. Therefore, the loss function value associated to one ranked list in a batch is computed as

$$\text{Listwise}_{KL(\mathcal{N}_{mult})} = D_{KL(\mathcal{N}_{mult})}(P||Q) * \boldsymbol{w} \tag{5.16}$$

where $P \sim \mathcal{N}(\mu_p, \Sigma)$ represents the output distribution of relevance probabilities returned by our model and $Q \sim \mathcal{N}(\mu_q, \Sigma)$ indicates the distribution of true relevance judgments associated to each item in a ranked list. The loss function values associated to each ranked list are then averaged over a batch.

Neural LETOR Model. The LETOR model that we consider in our experiments is depicted in Fig. 5.1. Its architecture is inspired to other recent neural models such as [34, 179, 166]. The proposed model first employs a standard Multi-Head Self-Attention Layer as the one introduced in [29] to compare the different document representations in input. Our model takes as input a batch of sets of items and returns the corresponding relevance scores to be used to rank them in decreasing order of relevance. In our case, the items to rank are the vector representations of different documents with respect to a user query. The documents are often represented by f real-valued features representing the matching score obtained by that document for the current query. These matching scores are computed with different retrieval model such as TF-IDF, BM25 or LM [81] taking in considerations different parts of the document such as the title, body or url [78, 79]. One document set at a time is then fed to a Multi Head Self-Attention Layer containing multiple Self-Attention heads like the ones described in [29], where the list of input documents is multiplied by three matrices of weights Q, K and V that are learned by the model and depicted in Fig. 5.3.

The outputs of each Self-Attention head – of size $m = f/k$, i.e. equal to the original document representation size f divided by the number k of attention heads used – are then concatenated and fed to a Regularization Layer with the goal of eliminating the redundancy in the representations of different attention heads and extracting only the features and their combinations which are useful for our goal. The Regularization Layer that we propose takes as input the concatenation of the outputs of different attention heads associated to the documents and begins by normalizing their components to have mean 0 and standard deviation equal to 1 (Layernorm). Then, we feed these normalized representations into a

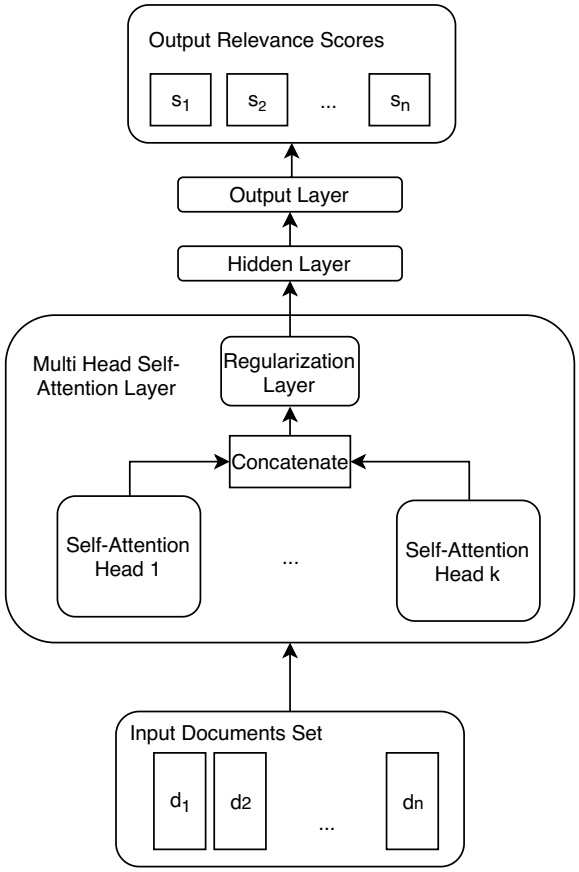


Fig. 5.1 Schema of the neural model employed in our experiments.

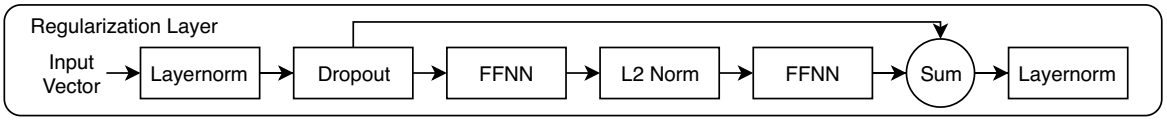


Fig. 5.2 Schema of the proposed Regularization Layer that we use in our LETOR model.

FFNN of size t times the input vector size f, using the ReLU activation function. Next, we normalize its output – with the goal of forcing to zero the components of the output which are redundant or not useful – and feed them again to another FFNN of size f. The output of the latter layer is then summed to the normalized input to the layer as described in Fig. 5.2. Finally, we normalize again the components of the output vector of this layer to have 0 mean and unit standard deviation. Each normalization layer estimates the components mean and standard deviations considering the components at the same index in each vector of a batch. This layer is used to improve the numerical stability of the operations in the model by maintaining the values within a constant range.

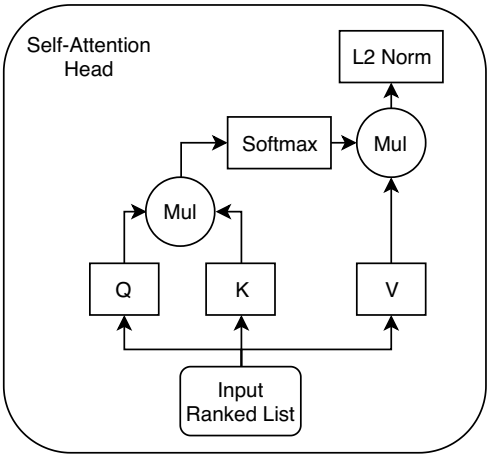


Fig. 5.3 Schema of the self-attention layer employed in our model. Q, K, and V are three matrices of the same size $(f \times m)$ that are learned by the model.

The output vector representation of each document returned by our Regularization Layer is finally fed to a hidden layer of size h with a ReLU activation function and to an output layer. The output layer size is equal to the number of relevance levels of the current collection if we are using the Pointwise$_{KL(Mul)}$ loss function, one otherwise. The activation function we use depends on the output size and is the $softmax$ function if we are considering a multi-class output and the *sigmoid* in the other cases.

5.8 Experimental Setup

In this section, we describe the considered experimental collections and the hyperparameters of the proposed LETOR model. We also motivate the evaluation measures we adopt and the experiments we report in Section 5.9. To ease reproducibility, additional technical details and the implementation of the proposed neural model and training functions are available online, along with the newly created test collection: `https://github.com/albpurpura/PLTR`.

Experimental Collections. The experimental collections that we consider in our experiments in this chapter are: MQ2007, MQ2008, MSLR-WEB30K [78] and OHSUMED [79], which are the main experimental collections of reference in the LETOR domain. All collections are already organized in five different folds with the respective training, test and validation subsets. We report the performance of our model averaged over these folds with the exception of the MSLR-WEB30K collection where we only considered Fold 1 since this is the most adopted choice among the approaches that employ this collection or its smaller subset MSLR-WEB10K [33, 172, 180, 166]. Additional details about all the experimental collections employed in this chapter are available in Section 2.8.

Crowdsourcing Relevance judgments. As mentioned above, we also conducted a crowdsourcing experiment to obtain a relevance judgments distribution for a subset of document-topic pairs from the COVID-19 MLIA collection. We consider the English MLIA subset, which contains 30 topics related to the COVID-19 pandemic and documents scraped from different online sources. [4] We collected relevance judgments on an average of 64 documents per topic (standard deviation = 4.37) with an average of 4 different judges (standard deviation = 7.12) per document. The judges are voluntary master or Ph.D. students in computer engineering or foreign languages. The relevance grades that we consider are in the $\{0,1,2\}$ set, with the usual interpretation. Next, we built a LETOR collection computing the documents features by employing different retrieval models and configurations available in the Apache Lucene framework. [5] We considered 24 different retrieval pipelines using a combination of one component from each the following sets: {BM25, Language Model with Dirichlet Smoothing (LMD), and Divergence From Randomness with Inverse Expected Document Frequency model with Bernoulli after-effect and normalization 2 (dfrinexpb2)} retrieval model, {Lucene, Indri, Atire, Okapi} stoplist – used as default options in the different search engines and available online [6] – and {Porter, Lovins} stemmer.

When required by the loss functions, we aggregated the relevance judgments distributions over 3 classes to obtain a real-valued relevance score. This was achieved by computing a weighted average with the following weights $[-1.0, 0.5, 1.0]$, obtaining a score within -1 and 1 which was then rescaled and shifted between 0 and 1 to be used for training and evaluation purposes. We chose this aggregation strategy as an alternative to the Majority Voting method, to better preserve all the possible degrees of relevance for a document-topic pair.

Model Hyperparameters and Training. The training parameters of the neural model are the number of attention heads k, the factor t which is used to determine the size $f \times t$ of the first FFNN in the proposed Regularization Layer, and the hidden layer size h. These parameters were adjusted according to the experimental collection size and number of features of the input data points and tuned on the validation sets of the first folds of each collection. The model was trained for a maximum of 100 epochs with early stopping with patience of 20 epochs on the MQ2007, MQ2008 and MSLR-WEB30K collections, while we reduced the maximum number of training epochs to 50 on the OHSUMED and MLIA collection. The best model was then selected for each fold according to the nDCG@1 performance on the validation set. The batch size was set to 4 on the MQ2007 collection, 2 on the MQ2008 and MLIA collections, 128 on the MSLR-WEB30K, and 1 on the OHSUMED

[4] `http://eval.covid19-mlia.eu`
[5] `https://lucene.apache.org`
[6] `https://bitbucket.org/frrncl/gopal/src/master/src/main/resources/`

collection. The number of attention heads k was dependent both on the collection size and the number of features available to represent each document. This parameter was set to 2 for the experiments on the MQ2007 and MQ2008 collections, 4 on the MSLR-WEB30K, 3 on the OHSUMED and 1 on the MLIA collection. The factor t was empirically set to 3 for the experiments on all collections, while the hidden layer size h was set to 32 for the experiments on the MQ2007, MQ2008 and OHSUMED collections, to 128 for the ones on the MSLR-WEB30K and to 8 for the experiments on MLIA.

Since in the MSLR-WEB30K and OHSUMED collections the maximum number of documents per query was much larger than the average, for efficiency reasons, we reranked only the top 150 documents ranked by the LightGBM LambdaMART model [181] tuned considering the nDCG@1 measure on the validation set of each collection fold. We cut the ranked lists at 150 documents since this was also the maximum value of documents available to rerank per-query on the MQ2007, MQ2008 and MLIA collections. The parameter α of the ApproxNDCG and ApproxNDCG with ST losses was set to 10 as in [21]. The proposed neural model was trained using the Adam optimizer with a learning rate of 0.001 on the MQ2007, MQ2008 and OHSUMED collection, of 0.0005 on the MSLR-WEB30K and of 0.0001 on the MLIA collection.

Evaluation Measures and Baselines. Because the main focus of LETOR models is to improve the quality of the top part of a ranked list, we evaluate the performance of the proposed loss functions relying on top-heavy widely-used measures as ERR [70], nDCG@{1, 3, 5} [7] and P@{1, 3, 5}; these are also amongst the most used measures in the LETOR literature. Moreover, despite the recent critics [182, 183] and the open debate in the community, to ease the comparison with other LETOR approaches in the literature [165, 180, 184], we also report the Average Precision (AP) of each of our runs averaged over all topics (MAP). We compute a paired Student's t-test and report the performance difference between the baseline of choice (i.e., the proposed neural model trained with the ApproxNDCG loss function or the LambdaMART model, depending on the experiment at hand) and the same model trained with the other loss functions; we indicate with $\uparrow$ or $\downarrow$ a statistically significant difference ($\alpha < 0.05$), accordingly to the sign of the difference. We selected the ApproxNDCG loss as a reference loss function for the proposed neural model since it was the one leading to the highest performance across all the considered collections compared to the other loss functions that we evaluated, i.e. MSE, Hinge, ApproxNDCG with ST, RankNet [61], ListMLE [185] and ApproxMRR [170]. To simplify the presentation of our evaluation, we only report the performance of our model when trained using the following loss functions as representatives

[7] The nDCG formulation that we employ is the same as the variant used by the popular `TREC_eval` tool. `https://github.com/usnistgov/trec_eval`

from the pointwise, pairwise and listwise loss categories: MSE, Hinge, ApproxNDCG with and without ST.

Given the success of the LambdaMART model in the LETOR domain and its widespread usage as a strong reference baseline for the evaluation of new systems [166], we also evaluate the impact of the proposed loss functions to train a decision tree-based GBM model, that is the model at the base of LambdaMART [22]. We only report the performance obtained by the GBM model with the top proposed loss functions for each collection. To perform the experiments, we rely on the LightGBM library. On each collection, we tune the hyperparameters of the GBM to obtain the highest nDCG@1 on the validation set of Fold 1 of each experimental collection.

The hyperparameter optimization process is performed – similarly to [78] – through a grid search over the following parameters: learning rate $\{0.001, 0.05, 0.1, 0.5\}$, number of trees $\{300, 500, 1000\}$, number of leaves $\{200, 500, 1000\}$ and followed by an additional manual tuning around the best hyperparameters combination found in the previous step. The best model hyperparameters for each collection are reported in the online repository. We also compare the performance of the neural LETOR and GBM model to the LightGBM LambdaMART implementation. We tuned its hyperparameters following the same criteria used for other models to obtain a reliable strong baseline [166]. We validated the newly proposed tuning strategy by comparing our results to the ones of recent state-of-the-art [186, 21, 19, 167, 166], obtaining comparable or better performances. Since providing a new state-of-the-art performance on the selected collections is not at the core of this research, and given the difficulty to find reliable performance reports of the wide variety of models available in the literature – many of them using different experimental setups or implementation of the evaluation measures (e.g., nDCG definition or weighting scheme and log base) – we consider our tuned LambdaMART model as a reference to position our results in the literature.

5.9 Evaluation

In Table 5.1, we report the performance of the proposed LETOR neural model trained using different loss functions on the MQ2007, MQ2008, MSLR-WEB30K and OHSUMED collections. We observe that the neural model achieves the best performance in the majority of the collections when trained with the Pairwise$_{KL(\mathcal{N})}$ loss function. On the MQ2007 and MQ2008 collections, all the proposed pairwise and listwise functions outperform the ApproxNDCG loss and most of the other losses with the exception of the MSE loss on the MQ2008 collection which is the most competitive amongst the baselines. On the MSLR-WEB30K collection the proposed Pairwise$_{KL(Bin)}$ loss performs as well as the Hinge loss

on this collection. We also observe that the performance of the model trained using the ApproxNDCG with ST loss is higher here compared to other collections. Indeed, on smaller collections the amount of training data is probably not sufficient for the model to benefit from this traning strategy. On the OHSUMED collection, the best loss function is the Listwise$_{KL(\mathcal{N}_{mult})}$, followed by the ApproxNDCG loss. This is due to the combination of a high proportion of relevant documents in the ranked lists and little amount of training data. For all the experiments on this collection, we rerank the top 150 documents returned by a LambdaMART model and, for this reason, we expect the representations of the documents in each ranked list to be more similar to each other rather than in other collections. Hence, approaches that consider multiple documents at a time, might have an advantage in finding the differences between them and providing more insightful information through their gradients to the model during training.

In Table 5.2 we report the percentage of topics where the proposed LETOR neural model using our probabilistic loss functions outperforms the previously considered baseline, i.e. the same LETOR neural model trained with the ApproxNDCG loss. In this case we observe that, despite the sometimes not-statistically significant performance improvements reported in Table 5.1, the proposed probabilistic loss functions provide a widespread performance increase on at least 50% of the topics on all performance measures – there are only two exceptions on the OHSUMED collection, where the AP achieved by the proposed pairwise loss functions only improves the AP of the model on 24-27% of the topics compared to the same model trained with the ApproxNDCG loss. However, on all collections, at least one of the proposed loss functions always achieves a performance improvement on more than 50% of the topics compared to the baseline model.

In Table 5.3, we report the results of the experiments training the proposed LETOR neural model using sampled relevance judgments. We sample the training labels from a Binomial distribution with parameters $n = 32$ – we determined this value empirically according of the performance of the model on the validation set of the first fold of the experimental collections – and p equal to the normalized relevance label found in the dataset. Then, we use the average of the sampled values for each document-topic pair as the training labels for our model for this whole set of experiments. This process does not change the relevance label of highly relevant or not-relevant documents – i.e. the documents with the highest or lowest relevance grade that have a success probability of 1.0 or 0.0, respectively – but provides some range of variability for the documents judged as *partially-relevant*, with a variance proportional to the uncertainty on the relevance of the document. Indeed the variance of each Binomial variable $P \sim \text{Bin}(n, p)$ is defined as $np(1-p)$, it is therefore higher for values closer to 0.5. This experiment allows us to assess whether there is a margin of improvement in the performance

of our model when adding some random noise to the labels of documents, which relevance is difficult to establish, simulating the annotation process of human judges.

As we can see in Table 5.3, the relative performance of the neural model when relying on different loss functions remains similar and the best loss function overall is still the Pairwise$_{KL(\mathcal{N})}$ loss. However, the neural model performance is often higher in this case than in the previous experimental setup, regardless of the loss function used. This is true for at least one performance measure when using all but the Pairwise$_{KL(\text{Bin})}$ loss on the MQ2007, and for all the proposed losses on the MQ2008 collection. We also observe a few performance improvements in the MSLR-WEB30K collection when using all loss functions with the exception of the ApproxNDCG with ST. Finally, on the OHSUMED collection, we observe a performance improvement in at least one measure when using the ApproxNDCG, ApproxNDCG with ST, MSE or Pairwise$_{KL(\text{Bin})}$ loss. These results support our hypothesis that acknowledging and exploiting the possible inconsistencies in the training data can be a viable way to improve a LETOR model's performance. The observed performance gains motivated us to further investigate on this aspect and to train our LETOR model on the raw relevance judgments distributions that we obtained from the crowdsourcing experiment on the COVID19-MLIA collection.

In Table 5.4, we report the experiments of the crowdsourcing experiments on the COVID19-MLIA collection. In this case, we set the output size of the neural model to 3 – the same number of relevance grades that we used for our annotation – and trained the model either (i) by aggregating – as explained in Section 5.8 – the model output probability scores and the collected relevance judgments distributions in the same way, obtaining one relevance score and relevance label to train the model with the previously evaluated loss functions, or (ii) by training the model with the proposed Pointwise$_{KL(Mul)}$ loss function which can take into account the raw probability distributions over the three relevance classes.

We observe that the proposed Pointwise$_{KL(Mul)}$ loss function is the best loss function to train the neural model on this collection. This can also be partially due to the small size of the collection which favors pointwise and pairwise loss functions. However, Pointwise$_{KL(Mul)}$ still outperforms other pointwise loss functions such as the MSE and the Pointwise$_{KL(Bin)}$ losses by a sizable margin. The Hinge loss is the second best loss function on this collection, outperforming all other baselines. The results from this experiment further confirm our initial hypothesis on the feasibility of training a LETOR model using raw probability distributions as training data.

Finally, in Table 5.5 we report the performance comparison between the best performing systems relying on the proposed neural model or on a decision tree-based GBM from the LightGBM library. We also report here the performance of a LambdaMART model trained

with the LigthGBM library as a baseline. LambdaMART, and in particular its LightGBM implementation is generally considered a very competitive baseline in many other LETOR research works [21, 20, 167, 19, 166]. We observe that the proposed probabilistic loss functions are in most of the cases cases able to improve the performance of a decision tree-based GBM model, surpassing the one of LambdaMART. We also observe that, on the MQ2007 and OHSUMED experimental collections, the proposed neural model outperforms LambdaMART and all the GBM-based models by a sizable margin.

To conclude our evaluation, in Table 5.6 we report a comparison of the performance improvements obtained using the proposed loss functions to train two state-of-the-art models, i.e. DASALC [166] and a simpler transformer model similar to the one used in [34]. More specifically, we report – for each model – the performance difference when training it using one of the proposed loss functions or the ApproxNDCG loss. From the results reported in Table 5.6, we observe that in the majority of our tests the proposed family of loss functions allows a better performance of both DASALC and the transformer model compared to the ApproxNDCG loss, especially on the MQ2008 collection. On the MQ2007 collection, we observe performance improvements on all evaluation measures, with some differences between the two models. In this case, the two best loss functions to train the DASALC model are the pairwise ones, while the transformer model benefits more from the pointwise and listwise losses. This effect is observed because of the simpler nature of the latter model and its lower number of parameters to train. On the MSLR-WEB30K collection, we observe fewer performance improvements compared to other datasets. However, the performance differences are more similar across models. Here, the best loss function is the Pairwise KL (Binomial), which leads to statistically significant performance improvements for both the considered neural LETOR models. Finally, on the OHSUMED collection we observe a significant performance improvement on almost all performance measures when training the transformer model with any of the proposed loss functions. On the other hand, the DASALC model shows fewer performance improvements, likely because of the higher model complexity combined with the small number of topics in this collection. Hence, the best loss functions in this case are the Listwise KL (Gaussian) and the Pointwise KL (Binomial), the same that performed the best on the proposed transformer-based neural LETOR model.

Observing all the above results, we notice that the performance of different models trained with the proposed loss functions varies according to the experimental collection used. Indeed, we conducted our evaluation selecting a number of datasets with different characteristics to show all the strengths and weaknesses of each of the proposed loss functions and to show the best scenarios where they can be employed. For example, for what concerns the

MQ2007 and M2008 collections – with 1,700 and 800 topics, respectively – we always ranked 128 (or less) documents for each topic. On the other hand, for the MSLR-WEB30K and OHSUMED collection – with 30,000 and 106 topic each – we considered a subset of size 128 (or smaller if fewer documents were available) of all documents provided for each topic in the dataset. We selected these subsets by ranking all the available documents for each topic with a LambdaMART model, and then then discarding the items with a rank higher than 128.

As a consequence of the diversity of the considered collections, we observe a few differences in the performance of the proposed loss functions in each of our experiments. On medium-sized collections – where documents were not filtered prior to ranking them – such as the MQ2007 and MQ2008, we observe overall a similar performance of all the proposed pointwise, pairwise and listwise loss functions, with sizeable differences noticeable only with certain evaluation metrics such as P@{1-5} and nDCG@{1-5}. On the MSLR-WEB30K dataset we observe a similar trend, here however we notice a more sizeable performance difference between the pairwise and listwise loss functions we propose and the pointwise variant – especially when using them to train the DASALC and the proposed LETOR model. In this case, the larger availability of training data and the document filtering step we applied before reranking contribute to the better performance of more complex training strategies – i.e. pairwise and listwise loss functions.

Finally, on the OHSUMED collection – the smallest of all the datasets we considered – we observe a different situation. Here, despite the small number of topics available, the best-performing loss function is the Listwise KL (Gaussian). This is likely due to the document pruning step we perform prior to ranking. This step promotes the selection of documents which are more similar to each other than in the previous cases and therefore gives an advantage to loss functions which compare multiple items at a time, i.e. the proposed listwise loss function. The second best performing probabilistic loss function however is the pointwise KL one. Indeed, this loss function allows a LETOR model to learn better than other training strategies when a few training examples are available – since it considers each document-topic pair as a valid training data-point, simplifying the training objective.

	Loss Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	ApproxNDCG	0.3169	0.4639	0.4291	0.4105	0.4152	0.4150	0.4219	0.4603
	ApproxNDCG (ST)	0.2704↓	0.3842↓	0.3621↓	0.3494↓	0.3363↓	0.3391↓	0.3459↓	0.4021↓
	Hinge	0.2674↓	0.3723↓	0.3633↓	0.3511↓	0.3239↓	0.3334↓	0.3420↓	0.4004↓
	MSE	0.3154	0.4574	0.4291	0.4115	0.4113	0.4136	0.4205	0.4550↓
	Pointwise KL (Binomial)	0.3168	0.4551	0.4334	0.4132	0.4087	0.4177	0.4230	0.4601
	Pairwise KL (Binomial)	0.3196	0.4728	0.4377	0.4186↑	0.4249	0.4226	0.4297↑	0.4647
	Pairwise KL (Gaussian)	**0.3218**	**0.4817**↑	**0.4381**	**0.4201**↑	**0.4350**↑	**0.4249**↑	**0.4318**↑	**0.4665**↑
	Listwise KL (Gaussian)	0.3177	0.4657	0.4332	0.4145	0.4173	0.4192	0.4255	0.4634
MQ2008	ApproxNDCG	0.2972	0.4222	0.3639	0.3337	0.3750	0.4039	0.4484	0.4585
	ApproxNDCG (ST)	0.2783↓	0.4005	0.3571	0.3171↓	0.3457↓	0.3856	0.4234↓	0.4373↓
	Hinge	0.2642↓	0.3533↓	0.3350↓	0.3099↓	0.3087↓	0.3548↓	0.4035↓	0.4228↓
	MSE	0.2997	**0.4401**	0.3844↑	0.3431↑	0.3795	0.4177↑	0.4596↑	0.4709↑
	Pointwise KL (Binomial)	0.2968	0.4222	0.3797↑	0.3375	0.3699	0.4126	0.4520	0.4626
	Pairwise KL (Binomial)	0.2995	0.4388	0.3797↑	0.3375	0.3839	0.4179	0.4567	0.4681↑
	Pairwise KL (Gaussian)	**0.3019**	0.4375	**0.3852**↑	0.3398	**0.3871**	**0.4222**↑	0.4603↑	0.4697↑
	Listwise KL (Gaussian)	0.3008	0.4349	0.3814↑	**0.3457**↑	0.3827	0.4171	**0.4630**↑	**0.4729**↑
MSLR-WEB30K	ApproxNDCG	0.3523	0.7504	0.7158	0.6936	0.5263	0.5090	0.5084	0.5798
	ApproxNDCG (ST)	0.3707↑	0.7804↑	0.6994↓	0.6811↓	0.5769↑	0.5065	0.5040↓	0.5818↑
	Hinge	**0.3748**↑	**0.7885**↑	0.6899↓	0.6745↓	**0.5884**↑	0.5054	0.5046	0.5877↑
	MSE	0.3623↑	0.7709↑	0.7334↑	0.7093↑	0.5461↑	**0.5268**↑	**0.5248**↑	0.5921↑
	Pointwise KL (Binomial)	0.3502	0.7596	0.7271↑	0.6985↑	0.5293	0.5136↑	0.5093	0.5862↑
	Pairwise KL (Binomial)	0.3467↓	0.7812↑	**0.7446**↑	**0.7184**↑	0.5357↑	0.5242↑	0.5241↑	**0.5971**↑
	Pairwise KL (Gaussian)	0.3454↓	0.7753↑	0.7423↑	0.7166↑	0.5315	0.5209↑	0.5214↑	**0.5971**↑
	Listwise KL (Gaussian)	0.3523	0.7612↑	0.7258↑	0.7016↑	0.5322	0.5152↑	0.5141↑	0.5871↑
OHSUMED	ApproxNDCG	0.4981	0.5660	0.5377	0.5057	0.4764	0.4544	0.4371	0.3828
	ApproxNDCG (ST)	0.4159↓	0.5000	0.4119↓	0.3660↓	0.4009	0.3330↓	0.3098↓	0.2882↓
	Hinge	0.4609	0.5660	0.4748↓	0.4566	0.4434	0.3930↓	0.3805↓	0.3530↓
	MSE	0.4376↓	0.5377	0.4465↓	0.4057↓	0.4340	0.3663↓	0.3417↓	0.3049↓
	Pointwise KL (Binomial)	0.4875	0.5189	0.5031	0.5019	0.4481	0.4269	0.4312	0.3800
	Pairwise KL (Binomial)	0.4206↓	0.4811	0.4151↓	0.3868↓	0.3915↓	0.3360↓	0.3235↓	0.2886↓
	Pairwise KL (Gaussian)	0.4520	0.5377	0.4403↓	0.4000↓	0.4481	0.3707↓	0.3481↓	0.2903↓
	Listwise KL (Gaussian)	**0.5248**	**0.6038**	**0.5692**	**0.5057**	**0.5189**	**0.4796**	**0.4456**	**0.3861**

Table 5.1 Performance of the proposed LETOR neural model averaged over all topics. ↑ or ↓ indicate a statistically significant difference ($\alpha < 0.05$) with the performance obtained using the ApproxNDCG loss function. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

	Loss Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	Pointwise KL (Binomial)	93%	83%	73%	59%	94%	89%	84%	59%
MQ2007	Pairwise KL (Binomial)	93%	81%	74%	62%	94%	88%	85%	61%
MQ2007	Pairwise KL (Gaussian)	94%	81%	74%	62%	95%	88%	84%	62%
MQ2007	Listwise KL (Gaussian)	94%	83%	75%	60%	95%	90%	85%	61%
MQ2008	Pointwise KL (Binomial)	91%	79%	73%	70%	92%	89%	88%	72%
MQ2008	Pairwise KL (Binomial)	93%	81%	75%	70%	95%	91%	89%	73%
MQ2008	Pairwise KL (Gaussian)	93%	82%	77%	72%	94%	92%	90%	74%
MQ2008	Listwise KL (Gaussian)	92%	83%	77%	73%	93%	91%	91%	75%
MSLR-WEB30K	Pointwise KL (Binomial)	85%	64%	55%	50%	93%	84%	78%	57%
MSLR-WEB30K	Pairwise KL (Binomial)	83%	65%	59%	54%	94%	87%	83%	68%
MSLR-WEB30K	Pairwise KL (Gaussian)	84%	65%	59%	54%	94%	87%	83%	68%
MSLR-WEB30K	Listwise KL (Gaussian)	84%	64%	58%	52%	93%	84%	79%	58%
OHSUMED	Pointwise KL (Binomial)	81%	63%	54%	50%	83%	72%	72%	52%
OHSUMED	Pairwise KL (Binomial)	81%	49%	36%	39%	83%	59%	50%	24%
OHSUMED	Pairwise KL (Gaussian)	83%	50%	39%	43%	86%	59%	51%	27%
OHSUMED	Listwise KL (Gaussian)	84%	66%	58%	49%	88%	80%	70%	55%

Table 5.2 Proportion of topics (%) where the performance of the proposed neural LETOR model using the proposed probabilistic loss functions is $\geq$ than the performance of the same model relying on the ApproxNDCG loss (on the same topic) according to different evaluation measures.

	Loss Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	ApproxNDCG	0.3175	0.4645	0.4342	0.4090	0.4125	0.4186	0.4217	0.4608
	ApproxNDCG (ST)	0.2749↓	0.3895↓	0.3690↓	0.3579↓	0.3398↓	0.3459↓	0.3540↓	0.4061↓
	Hinge	0.2667↓	0.3806↓	0.3599↓	0.3486↓	0.3295↓	0.3331↓	0.3416↓	0.4017↓
	MSE	0.3162	0.4569	0.4334	0.4117	0.4122	0.4167	0.4208	0.4554↓
	Pointwise KL (Binomial)	0.3185	0.4569	0.4297	0.4142	0.4119	0.4159	0.4246	0.4602
	Pairwise KL (Binomial)	0.3147	0.4592	0.4356	0.4158	0.4116	0.4163	0.4225	0.4612
	Pairwise KL (Gaussian)	**0.3193**	**0.4758**	**0.4383**	**0.4201**↑	**0.4288**↑	**0.4235**	**0.4294**↑	**0.4655**
	Listwise KL (Gaussian)	0.3187	0.4616	0.4370	0.4178↑	0.4146	0.4204	0.4272	0.4621
MQ2008	ApproxNDCG	0.2915	0.4120	0.3588	0.3286	0.3661	0.3951	0.4381	0.4498
	ApproxNDCG (ST)	0.2659↓	0.3890	0.3384↓	0.3077↓	0.3361	0.3676↓	0.4091↓	0.4297↓
	Hinge	0.2606↓	0.3457↓	0.3316↓	0.3092↓	0.3061↓	0.3521↓	0.4005↓	0.4194↓
	MSE	0.2991	0.4260	0.3835↑	0.3416↑	0.3744	0.4135↑	0.4568↑	0.4663↑
	Pointwise KL (Binomial)	0.2999	0.4311	0.3831↑	0.3378↑	0.3782	0.4193↑	0.4554↑	0.4671↑
	Pairwise KL (Binomial)	0.3016↑	0.4362↑	0.3814↑	0.3390↑	0.3871	0.4173↑	0.4584↑	0.4667↑
	Pairwise KL (Gaussian)	**0.3081**↑	**0.4503**↑	**0.3946**↑	**0.3449**↑	**0.4005**↑	**0.4314**↑	**0.4680**↑	**0.4769**↑
	Listwise KL (Gaussian)	0.2996	0.4311	0.3865↑	0.3439↑	0.3788	0.4210↑	0.4611↑	0.4717↑
MSLR-WEB30K	ApproxNDCG	0.3522	0.7523	0.7171	0.6938	0.5263	0.5091	0.5082	0.5797
	ApproxNDCG (ST)	0.1762↓	0.5103↓	0.5086↓	0.5060↓	0.2614↓	0.2773↓	0.2913↓	0.5041↓
	Hinge	0.3853↑	**0.7919**↑	0.7126	0.6998↑	**0.5926**↑	**0.5310**↑	**0.5316**↑	0.5887↑
	MSE	**0.3638**↑	0.7718↑	0.7344↑	0.7098↑	0.5483↑	0.5286↑	0.5262↑	0.5924↑
	Pointwise KL (Binomial)	0.3545	0.7542	0.7245↑	0.6986↑	0.5305	0.5137↑	0.5106	0.5883↑
	Pairwise KL (Binomial)	0.3475↓	0.7777↑	0.7406↑	0.7150↑	0.5352↑	0.5208↑	0.5212↑	0.5957↑
	Pairwise KL (Gaussian)	0.3506	0.7812↑	**0.7475**↑	**0.7202**↑	0.5402↑	0.5272↑	0.5267↑	**0.5988**↑
	Listwise KL (Gaussian)	0.3513	0.7529	0.7234↑	0.6987↑	0.5250	0.5129	0.5111	0.5870↑
OHSUMED	ApproxNDCG	0.5157	**0.6038**	**0.5377**	0.4906	**0.5189**	**0.4684**	0.4395	0.3729
	ApproxNDCG (ST)	0.4584↓	0.5660	0.4465↓	0.4208↓	0.4623	0.3765↓	0.3613↓	0.3257↓
	Hinge	0.4278↓	0.5094↓	0.4465↓	0.4358↓	0.4009↓	0.3604↓	0.3517↓	0.3475↓
	MSE	0.4474↓	0.5566	0.4528↓	0.4075↓	0.4434↓	0.3776↓	0.3493↓	0.3160↓
	Pointwise KL (Binomial)	0.4820	0.5094	0.5000	0.4830	0.4340	0.4244	0.4177	0.3727
	Pairwise KL (Binomial)	0.4203↓	0.4717↓	0.4182↓	0.3811↓	0.3915↓	0.3376↓	0.3185↓	0.2887↓
	Pairwise KL (Gaussian)	0.4446↓	0.5094↓	0.4340↓	0.3906↓	0.4245↓	0.3647↓	0.3401↓	0.2933↓
	Listwise KL (Gaussian)	**0.5165**	0.5755	**0.5377**	**0.5113**	0.5000	0.4654	**0.4473**	**0.3873**↑

Table 5.3 Performance of the proposed LETOR neural model averaged over all topics. The model is trained sampling relevance labels from a Binomial distribution. ↑ or ↓ indicate a statistically significant difference ($\alpha < 0.05$) with the ApproxNDCG baseline. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

	Loss Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
	ApproxNDCG	0.3313	0.4000	0.4667	0.4800	0.3556	0.3896	0.3973	0.3593
	ApproxNDCG (ST)	0.2544	0.3000	0.3333↓	0.3267↓	0.2708	0.2788	0.2820↓	0.3144
	Hinge	0.3455	**0.5000**	0.4667	0.4467	**0.4330**	0.3830	0.3736	0.3822
	MSE	0.2675	0.3333	0.3333	0.3333↓	0.2917	0.2894	0.2900↓	0.3101↓
MLIA	**Pointwise KL (Multinomial)**	**0.3377**	0.4333	**0.5000**	**0.4933**	0.3628	**0.4096**	**0.4054**	**0.3834**↑
	Pointwise KL (Binomial)	0.2562	0.3333	0.3222↓	0.3533↓	0.2798	0.2684↓	0.2898↓	0.3122
	Pairwise KL (Binomial)	0.2639	0.3667	0.3444↓	0.3200↓	0.3111	0.2919↓	0.2814↓	0.3023↓
	Pairwise KL (Gaussian)	0.2423↓	0.3333	0.3000↓	0.2867↓	0.2750	0.2515↓	0.2498↓	0.3062
	Listwise KL (Gaussian)	0.2521	0.3333	0.3111↓	0.3133↓	0.2715	0.2656↓	0.2653↓	0.3258

Table 5.4 Performance of the proposed LETOR neural model on the COVID19-MLIA collection averaged over all topics. ↑ or ↓ indicate a statistically significant difference ($\alpha < 0.05$) with the ApproxNDCG baseline. Best performance measures per collection are in bold as the loss function with the majority of best measures per collection.

	Loss Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	AP
MQ2007	GBM – LambdaMART	0.3211	0.4669	0.4397	0.4167	0.4217	0.4243	0.4285	0.4646
	GBM – Pointwise KL (Binomial)	**0.3233**	0.4752	0.4354	0.4167	0.4303	0.4207	0.4275	0.4591↓
	GBM – Listwise KL (Gaussian)	0.3219	0.4592	**0.4399**	0.4164	0.4152	0.4240	0.4267	0.4595
	NN – Pairwise KL (Gaussian)	0.3218	**0.4817**	0.4381	**0.4201**	**0.4350**	**0.4249**	**0.4318**	**0.4665**
	NN – Listwise KL (Gaussian)	0.3177	0.4657	0.4332	0.4145	0.4173	0.4192	0.4255	0.4634
MQ2008	GBM – LambdaMART	0.3045	0.4413	0.3869	0.3446	0.3858	0.4260	0.4664	0.4746
	GBM – Pointwise KL (Binomial)	**0.3072**	**0.4439**	**0.3941**	**0.3464**	**0.3935**	**0.4325**	**0.4690**	0.4771
	GBM – Listwise KL (Gaussian)	0.2991	0.4362	0.3852	0.3444	0.3801	0.4214	0.4633	**0.4775**
	NN – Pairwise KL (Gaussian)	0.3019	0.4375	0.3852	0.3398	0.3871	0.4222	0.4603	0.4697
	NN – Listwise KL (Gaussian)	0.3008	0.4349	0.3814	0.3457	0.3827	0.4171	0.4630	0.4729
MSLR-WEB30K	**GBM – LambdaMART**	**0.3955**	**0.7918**	0.7541	0.7288	**0.5925**	**0.5711**	**0.5670**	0.6299
	GBM – Pointwise KL (Binomial)	0.3550↓	0.7789↓	0.7503	0.7261	0.5435↓	0.5344↓	0.5343↓	0.6326↑
	GBM – Listwise KL (Gaussian)	0.3861↓	**0.7918**	**0.7565**	**0.7323**↑	0.5825↓	0.5647↓	0.5615↓	**0.6347**↑
	NN – Pairwise KL (Gaussian)	0.3454↓	0.7753↓	0.7423↓	0.7166↓	0.5315↓	0.5209↓	0.5214↓	0.5971↓
	NN – Listwise KL (Gaussian)	0.3523↓	0.7612↓	0.7258↓	0.7016↓	0.5322↓	0.5152↓	0.5141↓	0.5871↓
OHSUMED	GBM – LambdaMART	0.4704	0.5283	0.4874	0.4906	0.4387	0.3980	0.4037	0.4175
	GBM – Pointwise KL (Binomial)	0.5036	0.5755	0.5220	0.5000	0.4953	0.4474	0.4330	0.4210
	GBM – Listwise KL (Gaussian)	0.5139	0.5755	0.5314	**0.5151**	0.5000	0.4643↑	**0.4525**↑	**0.4243**
	NN – Pairwise KL (Gaussian)	0.4520	0.5377	0.4403	0.4000↓	0.4481	0.3707	0.3481↓	0.2903↓
	NN – Listwise KL (Gaussian)	**0.5248**	**0.6038**	**0.5692**↑	0.5057	**0.5189**	**0.4796**↑	0.4456	0.3861↓

Table 5.5 Performance of different LETOR models (decision tree-based Gradient Boosted Machine (GBM) model or the Neural Model (NM)) trained with the best-performing proposed loss functions averaged over all topics. ↑ or ↓ indicate a statistically significant ($\alpha < 0.05$) difference with the LambdaMART model trained on the original relevance judgments. Best performance measures per collection are in bold as the loss function with the most best measures per collection.

	Optimization Function	ERR	P@1	P@3	P@5	nDCG@1	nDCG@3	nDCG@5	MAP
MQ2007	DASALC – Pointwise KL (Binomial)	-0.0014	-0.0047	-0.0032	**+0.0032**	-0.0062	-0.0057	-0.0014	-0.0011
	DASALC – Pairwise KL (Binomial)	**+0.0016**	**+0.0065**	**+0.0041**	**+0.0067**↑	**+0.0059**	**+0.0027**	**+0.0050**	**+0.0060**↑
	DASALC – Pairwise KL (Gaussian)	**+0.0007**	**+0.0035**	**+0.0047**	**+0.0080**↑	**+0.0027**	**+0.0038**	**+0.0062**	**+0.0062**↑
	DASALC – Listwise KL (Gaussian)	-0.0015	-0.0089	-0.0006	**+0.0020**	-0.0080	-0.0014	**+0.0004**	-0.0002
	TRANSFORMER – Pointwise KL (Binomial)	**+0.0099**↑	**+0.0035**	**+0.0106**↑	-0.0007	**+0.0098**	**+0.0112**↑	**+0.0033**	-0.0005
	TRANSFORMER – Pairwise KL (Binomial)	**+0.0084**↑	**+0.0047**	**+0.0091**	**+0.0041**	**+0.0095**	**+0.0078**	**+0.0033**	**+0.0025**
	TRANSFORMER – Pairwise KL (Gaussian)	-0.0022	-0.0154	-0.0022	-0.0045	-0.0115	-0.0060	-0.0087	-0.0039
	TRANSFORMER – Listwise KL (Gaussian)	**+0.0015**	**+0.0024**	**+0.0002**	-0.0022	**+0.0038**	**+0.0007**	-0.0012	**+0.0001**
MQ2008	DASALC – Pointwise KL (Binomial)	**+0.0116**↑	**+0.0255**	**+0.0136**↑	**+0.0128**↑	**+0.0236**	**+0.0172**↑	**+0.0226**↑	**+0.0179**↑
	DASALC – Pairwise KL (Binomial)	**+0.0162**↑	**+0.0306**↑	**+0.0234**↑	**+0.0158**↑	**+0.0344**↑	**+0.0284**↑	**+0.0275**↑	**+0.0236**↑
	DASALC – Pairwise KL (Gaussian)	**+0.0114**↑	**+0.0230**	**+0.0170**↑	**+0.0089**↑	**+0.0249**↑	**+0.0210**↑	**+0.0201**↑	**+0.0181**↑
	DASALC – Listwise KL (Gaussian)	**+0.0160**↑	**+0.0268**↑	**+0.0238**↑	**+0.0171**↑	**+0.0268**↑	**+0.0277**↑	**+0.0279**↑	**+0.0224**↑
	TRANSFORMER – Pointwise KL (Binomial)	**+0.0119**↑	**+0.0242**	**+0.0162**↑	**+0.0056**	**+0.0198**	**+0.0231**↑	**+0.0170**↑	**+0.0146**↑
	TRANSFORMER – Pairwise KL (Binomial)	**+0.0129**↑	**+0.0293**↑	**+0.0174**↑	**+0.0074**	**+0.0281**↑	**+0.0235**↑	**+0.0178**↑	**+0.0143**↑
	TRANSFORMER – Pairwise KL (Gaussian)	**+0.0022**	**+0.0051**	**+0.0230**↑	**+0.0018**	-0.0064	**+0.0173**↑	**+0.0051**	**+0.0046**
	TRANSFORMER – Listwise KL (Gaussian)	**+0.0117**↑	**+0.0281**↑	**+0.0170**↑	**+0.0094**↑	**+0.0191**	**+0.0239**↑	**+0.0213**↑	**+0.0190**↑
MSLR-WEB30K	DASALC – Pointwise KL (Binomial)	-0.0083↓	-0.0262↓	-0.0152↓	-0.0144↓	-0.0192↓	-0.0175↓	-0.0173↓	-0.0023↓
	DASALC – Pairwise KL (Binomial)	-0.0120↓	**+0.0165**↑	**+0.0209**↑	**+0.0216**↑	**+0.0028**	**+0.0062**↑	**+0.0077**↑	**+0.0158**↑
	DASALC – Pairwise KL (Gaussian)	-0.0169↓	**+0.0094**↑	**+0.0168**↑	**+0.0182**↑	-0.0074	-0.0009	**+0.0020**	**+0.0148**↑
	DASALC – Listwise KL (Gaussian)	-0.0067↓	-0.0263↓	-0.0175↓	-0.0138↓	-0.0163↓	-0.0139↓	-0.0125↓	-0.0039↓
	TRANSFORMER – Pointwise KL (Binomial)	-0.0238↓	**+0.0138**↑	-0.0028	-0.0146↓	-0.0099↓	-0.0193↓	-0.0243↓	-0.0082↓
	TRANSFORMER – Pairwise KL (Binomial)	-0.0440↓	**+0.0135**↑	**+0.0142**↑	**+0.0101**↑	-0.0386↓	-0.0235↓	-0.0197↓	**+0.0042**↑
	TRANSFORMER – Pairwise KL (Gaussian)	-0.0419↓	-0.0041	-0.0001	-0.0003	-0.0416↓	-0.0336↓	-0.0279↓	**+0.0019**↑
	TRANSFORMER – Listwise KL (Gaussian)	-0.1000↓	-0.0798↓	-0.0826↓	-0.0786↓	-0.1242↓	-0.1136↓	-0.1068↓	-0.0328↓
OHSUMED	DASALC – Pointwise KL (Binomial)	**+0.0232**	0.0000	**+0.0220**	**+0.0208**	**+0.0189**	**+0.0350**	**+0.0354**	**+0.0328**↑
	DASALC – Pairwise KL (Binomial)	**+0.0022**	**+0.0094**	-0.0346	-0.0547↓	**+0.0283**	-0.0141	-0.0260	-0.0107
	DASALC – Pairwise KL (Gaussian)	-0.0176	-0.0189	-0.0629	-0.0717↓	**+0.0047**	-0.0408	-0.0480↓	-0.0140
	DASALC – Listwise KL (Gaussian)	**+0.0266**	**+0.0472**	-0.0094	-0.0189	**+0.0566**	**+0.0178**	**+0.0050**	**+0.0026**
	TRANSFORMER – Pointwise KL (Binomial)	**+0.1563**↑	**+0.1321**↑	**+0.1478**↑	**+0.1434**↑	**+0.1745**↑	**+0.1604**↑	**+0.1573**↑	**+0.0825**↑
	TRANSFORMER – Pairwise KL (Binomial)	**+0.0530**	-0.0189	**+0.0409**	**+0.0434**	**+0.0283**	**+0.0477**	**+0.0474**	**+0.0393**↑
	TRANSFORMER – Pairwise KL (Gaussian)	**+0.0845**↑	**+0.0755**	**+0.0786**	**+0.0302**	**+0.1038**	**+0.0909**	**+0.0673**	**+0.0298**
	TRANSFORMER – Listwise KL (Gaussian)	**+0.1485**↑	**+0.1132**	**+0.1604**↑	**+0.1358**↑	**+0.1604**↑	**+0.1559**↑	**+0.1441**↑	**+0.0768**↑

Table 5.6 Performance of different LETOR models trained with the proposed loss functions. We indicate with ↑ or ↓ a statistically significant (p-value < 0.05) difference with the performance obtained by the same model trained with the ApproxNDCG loss function on the original relevance judgements available in each dataset. We indicate in bold all the cases where we observe a performance improvement over the respective baseline.

5.10 Final Remarks

In this chapter, we presented different strategies to train a LETOR model relying on relevance judgments distributions. We introduced five different loss functions relying on KL divergence between distributions, opening new possibilities for the training of LETOR models. The proposed loss functions were evaluated on a newly proposed neural model, two state-of-the-art transformer-based LETOR architectures and on a decision tree-based GBM model – the same model employed by the popular LambdaMART algorithm [22] – over a number of experimental collections of different sizes, i.e. MQ2007, MQ2008, MSLR-WEB30K and OHSUMED.

We compared the performance of the proposed loss functions to a selection of the most representative loss functions in the IR domain: the pointwise Mean Squared Error (MSE) loss [169], the pairwise Hinge loss [15] and the listwise ApproxNDCG loss [170] with and without the Stochastic Treatment (ST) proposed in [19]. In our experiments, the proposed loss functions outperformed the aforementioned baselines in several cases and gave a significant performance boost to LETOR approaches – especially the ones based on neural models – allowing them to also outperform other strong baselines in the LETOR domain such as the LightGBM implementation of LambdaMART [22, 166].

We also evaluated the option of training a neural LETOR model simulating the distribution of relevance judgments for each document-topic pair in the training data. The results from this experiment further confirmed our hypothesis on the utility of using relevance judgments distributions to train a LETOR model, showing performance improvements across different measures.

Finally, we conducted crowdsourcing experiments on the COVID-19 MLIA collection, building a new LETOR collection with real relevance judgments distributions. We share this collection and labels to be used for the development and evaluation of other LETOR approaches that will follow the proposed training strategies. These experiments consolidated our hypothesis and showed encouraging results on the usage of probabilistic loss functions also on this dataset.

Chapter 6

Improving the Efficiency of Neural Rankers

6.1 Introduction

As we argued in the previous chapters, one of the main advantages of LETOR models compared to NIR ones is their efficiency. By receiving pre-computed numerical features, LETOR models provide a scalable alternative for estimating the relevance of a query-document pair.

In the past few years, with the rise in popularity of deep learning, more LETOR systems have started employing these modeling strategies, often achieving a performance improvement over traditional ML models such as RFs. However, this performance improvement is often achieved at the expense of increasing the number of parameters, complexity and running time of such systems.

For this reason, we focus in this chapter on proposing an easily applicable strategy to improve the efficiency of neural LETOR models, without sacrificing their performance.

6.2 Research Question

In the recent years, the attention on neural approaches for this task has grown proportionally to their performance. Starting from [187], where the authors propose to employ a recurrent neural layer to model documents list-wise interactions, to [34], where the now popular self-attention transformer architecture is used. Also, the performance of neural models [34, 33] recently became competitive with approaches such as the LambdaMART [22] model which is often one of the first choices for LETOR tasks. However, neural models performance grew

at the expense of their complexity and this delayed their application in large-scale search systems.

Indeed, in such context, model latency and update time are as important as model performance. Especially since LETOR models are the most popular choice to refine a ranking in industrial applications. Once deployed, a LETOR model is expected to efficiently perform its task within the current pipeline as one of its building blocks with minimal maintenance. For these reasons, we focus in this chapter on the following research question.

RQ4: How can we improve the efficiency of a neural LETOR model?

Within this context, we believe feature selection approaches are the best solution to improve the efficiency of LETOR models, especially the ones which have already been deployed in a search pipeline. A feature selection system allows to reduce the number of features used to represent each query-document pair when it is given in input to LETOR systems. In this way, we only have to adjust the input size of the the current LETOR model and retrain it to achieve an efficiency improvement in the entire search pipeline. In addition to this, we can also reduce the overall cost of the pipeline, skipping the computation of features which are deemed as redundant by our feature selection approach. Overall, reducing the input size can help decreasing model architectural complexity, number of parameters, and consequently training and inference time. Moreover, previous works [188–190] already showed that the document representations used for LETOR can sometimes be redundant in non-trivial ways and often reduced [188] without impacting the ranking performance.

6.3 Methods

Our approach relies on the computation of saliency maps to compute the relative importance of each feature in a LETOR model's input. We then apply some data mining techniques that leverage this information to prune the set of input features. We evaluate the proposed technique on different publicly available collections described in Section 2.8 and compare it to state-of-the-art feature selection approaches from this domain. In particular, we reimplemented the two best-performing approaches proposed in [188] and consider them as our baselines: eXtended naive Greedy search Algorithm for feature Selection (XGAS) – which relies on LambdaMART to estimate feature relevance – and Hierarchical agglomerative Clustering Algorithm for feature Selection (HCAS) employing single likage [191] – which relies on Spearman's correlation coefficient between feature pairs as a proxy for feature importance. To the best of our knowledge, our approach is the first feature selection technique for LETOR specifically targeted to neural models.

6.4 Contributions

The main contributions of this paper are the following:

- we propose an architecture-agnostic Neural Feature Selection (NFS) approach which uses a neural LETOR model to estimate feature importance;
- we evaluate the quality of our approach on two public LETOR collections;
- we confirm the robustness of the extracted feature set evaluating the performance of the proposed neural reranker and of a LambdaMART model using subsets of features of different sizes computed with the proposed approach.

Our experimental results show that the document representations used for LETOR can sometimes be redundant and reduced to up to 40% [188, 38] of the total without impacting the ranking performance.

6.5 Outline

This chapter is organized as follows. In Section 6.6 we present the details of previously proposed feature selection approaches for LETOR models; in Section 6.7 we describe the proposed approach; in Section 6.9 we report the experimental setup used for our evaluation; in Section 6.9 we present the results of our evaluation. Finally, in Section 6.10 we present our final remarks on the proposed solution.

6.6 Related Work

Existing feature selection approaches can be organized into three main groups: *filter*, *embedded*, and *wrapper* methods [188].[1] Filter methods, such as the Greedy Search Algorithm (GAS) [189], compute one score for each feature – independently from the LETOR model that is going to be used afterwards – and select the top ones according to it. In GAS the authors minimize feature similarity (Kendall Tau) and maximize feature importance. They rank the input items using only one of the features at a time and consider as importance score the MAP or nDCG@k value. Embedded approaches, such as the one presented in [192], incorporate the feature selection process in the model. In [192], the authors propose to apply

[1] We purposely omit a comparison with other dimensionality reduction approaches such as PCA since these methods often compute a *combination* of the features to reduce the representation size which is beyond the scope of this work

different types of regularizations – such as L1 norm regularization – on the weights of a neural LETOR model to reduce redundancy in the hidden representations of the model and improve its performance. Finally, wrapper methods such as the ones presented in [188] and the proposed approach, rely on a LETOR model to estimate feature importance and then perform a selection.

6.7 Proposed Approach

The proposed Neural Feature Selection (NFS) approach is organized in the following three steps. We first train a neural model for the LETOR task, i.e. to compute a relevance score for each item in the input set to be used to rank it. Second, we use the trained model to extract the most significant features groups considered by the model to rank each item. Finally, we perform feature selection using the previously computed feature information. The algorithm we employ is formalized in the pseudocode below.

Algorithm 1: Neural Feature Selection(neuralLetorModel, trainData, devData, targetNFeatures)

```
neuralLetorModel.train(trainData)
saliencyMaps = neuralLetorModel.computeSaliencyMaps(devData)
featureGroupPatterns = extractFeatureGroups(saliencyMaps)
featureCorrelationMatrix = computeFeatureCorrMatrix(featureGroupPatterns)
featureClusters = hierarchicalClustering(featureCorrelationMatrix,
  targetNClusters=targetNFeatures)
selectedFeatures = []
for cluster ∈ featureClusters do
    f = selectMostFreqFromCluster(cluster, featureGroupPatterns)
    selectedFeatures.append(f)
end
return selectedFeatures
```

Neural Model Training. The NFS model architecture is composed of n self-attention layers [29], followed by two fully-connected layers. We train this model using the Approx-NDCG loss [21]. Before feeding the document vectors to the self-attention layer we apply the same feature transformation strategy described in [33]. In [33], the authors apply three different feature transformations to each feature in the input data and then combine them through a weighted sum. The weights for each transformation are learned by the model so that the best feature transformation strategy for each feature could be used each time. The model architecture is depicted in Figure 6.1. Also, we apply batch normalization to the input

of each feed-forward layer and dropout on the output of each hidden layer. Note that, since our approach for feature selection is *architecture-agnostic*, we can easily make changes to this neural architecture without impacting the following steps for feature selection.

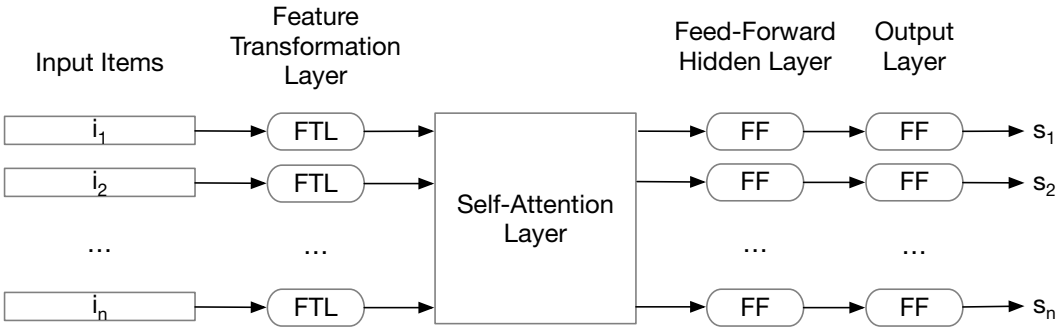


Fig. 6.1 Architecture of the neural model employed in our evaluation.

Feature Groups Mining. At this step, we use the model trained in the previous step to select the most important features used to rank each item in our training data. To do so, we compute the saliency map – a popular approach in the computer vision field to understand model predictions [193–195] – i.e. the gradient w.r.t. the each input item feature, corresponding to each item in the training dataset. We then apply min-max normalization on each saliency map M_i to map the values in each vector to the same range $[0,1]$. Afterwards, we select from each saliency map the groups of features g which have a *saliency* score higher than a threshold t. The set of feature groups G extracted at this step are the most significant features sets that our neural model learned to rely on to compute the relevance score of each item. These features however might not be the same for any possible input instance and – as also pointed out in [193] – saliency maps can often be noisy and not always represent the behavior of a neural model. For this reason, we propose to apply a further selection step to prune less reliable feature groups similarly to what proposed in [196] where the authors compute the statistical significance of groups of items by comparing their frequency of occurrence in real data to the one in randomly generated datasets. We compute K random sets of saliency maps, each of the same cardinality of the experimental dataset employed. For example, if a dataset contains N queries, each with R documents to be ranked, then we will generate K random datasets, each containing $N \times R$ saliency maps. Then, we apply the same feature groups extraction process on the random saliency maps and compute K different sets of feature groups. The saliency maps are computed sampling values from a uniform distribution with support $[0,1]$. According to this modeling strategy, each feature can be considered as salient in the current random saliency map with probability $1-t$; where t is

the threshold we used in the previous step to select salient features. Once we computed these K sets of random feature groups $\hat{G}_k$ we use their frequency to prune the original ones. In particular, we consider the frequency f_{g_i} of group $g_i \in G$ and compare it to its frequency in each of the K random datasets $f_{g_{i,k}}$ – the frequency $f_{g_{i,k}}$ might also be 0 if the feature group g_i does not appear in the random dataset k. If $f_{g_i} \leq f_{g_{i,k}}$ in more than 2% [2] of the randomly generated feature groups $\hat{G}_k$, we discard feature group g_i, considering it as noise.

Feature Selection. In this final step, we rely on the feature groups extracted in the previous step and their frequency in the saliency maps to compute a feature similarity matrix. We then use this similarity matrix to perform feature selection. Each feature pair similarity value is computed counting the times the two features appear in the same feature group and normalizing that score by the total number of groups where that feature appears. Finally, we rely on this similarity matrix to perform hierarchical clustering as done in [188]. We consider the number of clusters as the stopping criterion for the single linkage hierarchical clustering algorithm. The final set of features to keep is computed selecting the most frequently occurring feature in the previously computed feature groups, from each feature cluster.

6.8 Experimental Setup

We evaluate our approach on the first fold of the MSLR-WEB30K [197] and on the whole OHSUMED [79] dataset where the items to rank are represented by 136 and 45 features, respectively. [3] We use the LambdaMART implementation available in the LightGBM [4] library [181] and train and test the proposed neural model considering only the top 128 results returned by LambdaMART. We tuned the LightGBM model parameters on the validation sets of both datasets, optimizing the ndcg@3 metric.[5] The proposed neural reranking model is trained for 500 epochs – 100 epochs on the OHSUMED dataset – with batch size 128, using Adam optimizer and a learning rate 0.0005. We consider a feature embedding size of 128 in the feature transformation layer on the MSLR-WEB30K dataset – while we removed it for the experiments on the OHSUMED collection due to its much smaller size and number of features which limited the benefits of it – 4 self-attention heads on the MSLR-WEB30K and 1 on the OHSUMED dataset and a hidden size of 128 for the hidden feed-forward layer.

[2] This value was set empirically to yield a reasonable number of feature groups for the following feature extraction step

[3] `https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval`

[4] `https://github.com/microsoft/LightGBM`

[5] We set the learning rate to 0.05, the number of leaves to 200 and the number of trees to 1000 (500) on the MSLR-WEB30K (OHSUMED) collection

Since each attention head has an output size equal to the total number of features divided by the number of attention heads, to compute the results reported in Table 6.1, we reduce the number of attention heads to 1 when using 5% and 10% of all the available features (6 and 13 features respectively), we use 4 attention heads when considering 30% (27 features), and 3 when using 40% (54 features). The batch normalization momentum we use is 0.4 and the dropout probability is $p = 0.5$. In the feature groups mining step, we generate 5000 random datasets and the threshold t to extract the feature groups is empirically set to 0.95. For the evaluation of the approach we consider the nDCG@3 measure, similar results are obtained with nDCG at different cutoffs.

6.9 Experimental Results

In Table 6.1, we report the results of our experiments on the MSLR-WEB30K dataset. We trained both a LambdaMART model and the proposed neural reranking one on different subsets of features of increasing size. From these experiments, we observe that the proposed Neural Feature Selection (NFS) approach always outperforms all the other baselines when the selected features are used to train a LambdaMART model, and in most of the cases when used with the proposed neural model.

	LambdaMART			Neural Reranker		
Features Perc.	XGAS	HCAS (single)	NFS	XGAS	HCAS (single)	NFS
5%	0.358	0.359	**0.375**	**0.377**	0.360	0.375
10%	0.370	0.404	**0.420**	0.383	0.392	**0.412**
20%	0.378	**0.467**	**0.467**	0.383	**0.444**	0.443
30%	0.417	0.466	**0.471**	0.409	**0.448**	0.424
40%	0.439	0.471	**0.473**	0.394	0.452	**0.456**
100%	0.473	0.473	0.473	0.453	0.453	0.453

Table 6.1 Evaluation of the proposed Neural Feature Selection (NFS) approach on the MSLR-WEB30K dataset. We report the ndcg@3 values obtained by LambdaMART and the proposed Neural Reranking model employing different subsets of features.

The evaluation results on the OHSUMED dataset reported in Table 6.2 are computed as the previous case. Here, we consider 60%, 70%, 80%, and 90% of the total features in the collection since the total number of feature is much smaller than in the previous dataset. In our evaluation, NFS outperforms HCAS in the majority of the cases, even though the latter approach is slightly more competitive than before.

	LambdaMART			Neural Reranker		
Features Perc.	XGAS	HCAS (single)	NFS	XGAS	HCAS (single)	NFS
60%	0.367	0.378	**0.395**	0.421	**0.428**	0.424
70%	0.367	0.378	**0.386**	0.424	0.441	**0.444**
80%	0.367	0.399	**0.401**	0.437	**0.437**	0.421
90%	0.367	**0.405**	0.396	0.367	0.405	**0.422**
100%	0.397	0.397	0.397	0.497	0.497	0.497

Table 6.2 Evaluation of the proposed Neural Feature Selection (NFS) approach on the OHSUMED dataset. We report the ndcg@3 values obtained by LambdaMART and the proposed Neural Reranking model employing different subsets of features.

In Table 6.3 we report the performance difference (nDCG@3) on the MSLR-WEB30K and OHSUMED dataset between a LambdaMART or the proposed neural LETOR model using z% of the total features to represent an input query-document pair and the respective model using of 100% of them. The percentage of features z is selected on each dataset/model pair as the minimum percentage of features we can use on each dataset/model pair without worsening its original performance or observing the least performance degradation. This value corresponds to 40% of the total number of features for the LambdaMART and neural LETOR model on the MSLR-WEB30K dataset – since this is the minimum percentage of features we can employ to represent the inputs to both models before we observe a performance degradation. On the OHSUMED collection the same percentage z is equal to 80% for the LambdaMART model and 70% for the neural LETOR model. In this situation we observe that the proposed NFS approach is the best solution to maintain efficiency while observing a minimum performance degradation or, in most cases, even a performance improvement.

		XGAS	HCAS (single)	NFS
MSLR-WEB30K	LambdaMART	-0.034	-0.002	**+0.001**
	Neural Reranker	-0.058	-0.001	**+0.003**
OHSUMED	LambdaMART	-0.030	+0.002	**+0.004**
	Neural Reranker	-0.073	-0.056	**-0.053**

Table 6.3 Performance difference (nDCG@3) on the MSLR-WEB30K and OHSUMED dataset between a LambdaMART and the proposed neural LETOR model using z% of the total features to represent an input query-document pair instead of 100% of them.

The main advantage of using a subset of features to represent the inputs to a neural model is that we can reduce the model complexity. We observe this effect mainly when our data is

represented by a large number of features as in the MSLR-WEB30K collection. For example, when using 40% of the features of the dataset, the number of attention heads in our model was reduced from 4 to 3 and, since we were considering only 54 out of 136 features, the number of parameters of the self-attention heads – the first layer of our model – was also reduced. As a consequence, training time was halved and inference time also decreased.

It is also interesting to observe the differences between the features selected by the proposed NFS approach and other baselines. We focus on the top 3 features selected from the OHSUMED collection by each of the considered feature selection algorithms over the 5 different dataset folds and refer the reader to [79] for a more detailed description of each feature. NFS most frequently selected features computed with popular retrieval models such as BM25 or QLM [3] (features 4, 12 and 28) based on the document abstract or title. On the other hand, HCAS selected simpler features derived from raw frequency counts of the query terms in each document's title and abstract (features 23, 40 and 36). Finally, XGAS selected a mix of features computed with traditional retrieval approaches such as QL, and simpler frequency counts (features 2, 44 and 13). We conclude that the advantage of NFS is likely due to its ability to recognize and select the most sophisticated and useful matching scores thanks to the information learned during training.

6.10 Final Remarks

In recent years, neural models became a competitive alternative to traditional LETOR approaches. However, their performance grew at the expense of their efficiency and complexity. In this chapter, we proposed an approach for feature selection for LETOR based on a neural ranker. Our approach is specifically designed to optimize the performance of neural LETOR models without the need to change their architecture. In our experiments, the proposed approach improved the efficiency of a sample neural LETOR model and decreased its training time without significantly impacting its performance. We also validated the robustness of the selected features testing them using a different – non neural – model such as LambdaMART. We performed our evaluation on two popular LETOR datasets – i.e. MSLR-WEB30K and OHSUMED – comparing our approach to two state-of-the-art techniques from [188]. The proposed approach outperformed the selected baselines in the majority of the experiments on both datasets.

Chapter 7

Conclusion

Deep learning models revolutionized the research landscape in many fields of Computer Science, including *Information Retrieval (IR)*. They are now vastly employed to compare and analyze textual content and to process numerical data. However, they are generally used as a black-box component in most IR pipelines, without much awareness of the factors inside and outside of them that could affect their effectiveness.

In this thesis we aimed at filling this knowledge gap and improving the performance of these approaches when used in two popular contexts of ad-hoc retrieval, i.e. *Neural Information Retrieval (NIR)* and *LEarning TO Rank (LETOR)*.

First, we asked ourselves how we could improve our understanding of NIR models and identify the key factors that could affect their performance in a retrieval pipeline. To achieve this goal, we conducted – in **Chapter 3** – a reproducibility study of two popular NIR models, i.e. *Deep Relevance Matching Model (DRMM)* [15] and *Neural Vector Space Model (NVSM)*[13]. Our study allowed us to thoroughly assess all factors that could affect the performance of such models and to compile a list of guidelines for NIR researchers to follow in order to guarantee the reproducibility of their approaches. We also highlighted with an extensive evaluation of these models on a number of multi-language collections from different domains the scenarios where NIR models consistently succeed in outperforming existing lexical models. Finally, we focused on measuring the impact of different text representation models on the performance of these approaches. Indeed, a large number of NIR models rely on a language model to represent the semantic meaning of words. These models promise a reliable way to compute term similarity and to extract text-matching signals that can later be employed by NIR systems to estimate the relevance of a document with respect to a user query. Over the years, different *Word Embeddings (WE)* models have been proposed, and NIR systems have employed them as an interchangeable resource. Our work

showed how the choice of the word representation model is crucial in determining the success of NIR systems, especially when non-contextual representations of words are used.

Following this lead, in **Chapter 4** we investigated the potential of probabilistic word representations in IR, in particular, Elliptical *Probabilistic Word Embeddings (PWE)* [36]. This WE model was never adopted before in the IR domain, however, it is one of the many approaches proposed within the NLP community to encode the meaning of terms as numerical representations. More specifically, this model showed a significant potential to represent relations between terms, in particular hypernymy and hyponymy. In linguistics, two words in a hypernymy relation are for example "animal" and "dog" where every dog (hyponym) can be also defined as an animal (hypernym) but not vice versa. The main difference between PWE and other embedding models such as Word2Vec is their strategy to represent words not just with a real-valued vector – identifying a point in the multi-dimensional space – but also with a covariance matrix – identifying an elliptical portion of space – which can be used to encode additional information on a word's meaning. We assessed the impact of PWE in IR employing them as the word representation solution for a popular NIR model, i.e. MatchPyramid [14]. We then compared its performance relying on PWE or other representation strategies. We also proposed a variant of the MatchPyramid NIR architecture which allows it to employ graph-based *Word Embeddings (WE)*, without incurring in the frequent *Out-Of-Vocabulary (OOV)* problem. Our results confirm the potential of probabilistic WE models also in the IR field and their limitations, including some practical lessons we learned when building our evaluation pipeline.

Continuing our research work on deep learning models applied to IR, we considered next the LETOR domain. In this context, deep learning models receive an already pre-processed representation of the contents of a user query and documents in a collection. For this reason, such systems are *shallower* and more efficient than NIR ones and focus only on the core tasks of IR, i.e. relevance estimation and ranking.

In particular, we focus on a crucial aspect of LETOR which is model training and optimization. The development of new loss functions for training *Machine Learning (ML)* models to perform the ranking task has been the core of research in the LETOR domain since the early 2010s. In particular, in **Chapter 5** we proposed and evaluated a new family of training strategies for neural LETOR models that employ relevance judgments distributions instead of single-valued relevance labels. The problem of estimating relevance and interpreting training data is central in deep learning. More specifically, in the IR domain lots of work has been done on designing reliable relevance judgments collection strategies to build high-quality datasets for the evaluation of such systems. However, most of these approaches accepted the idea that a single relevance label provided by a judge or through some

data aggregation strategy is a reliable representation of the relevance of a query-document pair. In Chapter 5 we evaluated a different option, i.e. training a LETOR model relying on relevance judgments distributions. We proposed and implemented our theoretical framework to train a model using this data and evaluate the results on a popular LETOR model – i.e., LambdaMART [22] – and on a sample neural LETOR architecture that we propose. Our training strategies allowed both ML models to achieve a better performance in the ranking task, often in a statistically significant way.

Finally, in **Chapter 6** we propose a solution to improve the efficiency of neural LETOR models without sacrificing their performance. Indeed, in IR as in many other applied computer science fields, a model's response time is crucial for its widespread adoption. With the increased availability of larger training datasets and more sophisticated training strategies, the complexity of LETOR models also increased. In particular, neural LETOR models are easily affected by efficiency limitations because of their increasingly complex architectures. For this reason, we proposed in Chapter 6 an architecture-agnostic solution to prune the input representations of a neural LETOR model by automatically estimating their relative importance to the model itself. We achieve this relying on techniques from the Computer Vision and Data Mining fields such as saliency maps and pattern significance mining. According to our evaluation, the proposed feature selection approach allowed to reduce training and inference time of a sample neural LETOR model by up to 50% by reducing the size of query-document pairs' representations, without a sizable performance degradation.

Overall, our work presented in this thesis explored two key aspects for the advancement of research in this field and for the successful application of deep learning models in the IR field. We first studied how to improve the presentation and design of an NIR model, taking into account the impact of each of its components. We believe our work in this context provided important insights for the reproducibility of research works in this field and advanced the understanding of the role of each of the main components of a NIR system. Next, we proposed new training strategies to train a reranking model and proposed a solution to improve its efficiency. Our research work in this direction led to the development of a new family of probabilistic loss functions that consistently improves the performance ofLETOR models and to an algorithm for feature selection that highly improves the efficiency of neural LETOR models.

Future research works building on our work could (i) provide a systematic analysis of the interactions between the components of NIR systems that we identified and analyzed independently in Chapter 3 using evaluation frameworks like *ANalysis Of VAriance (ANOVA)* [198]; (ii) improve the efficiency and study new ways to embed contextual information in the proba-

bilistic embeddings we studied in Chapter 4, including for example this representation in a more lightweight transformer-based model; (iii) evaluate how the proposed probabilistic loss functions would apply to larger-scale LETOR models or NIR approaches; (iv) compare knowledge distillation approaches to the proposed feature selection solution and evaluate the advantages in terms of performance and execution time.

We hope our work will help future researchers in the deep learning and IR domain to improve the quality of modern search systems and to simplify and expand access to information for everyone.

References

[1] E. A. Fox and J. Shaw. Combination of Multiple Searches. In D. K. Harman, editor, *The Second Text REtrieval Conference (TREC-2)*, pages 243–252. National Institute of Standards and Technology (NIST), Special Pubblication 500-215, Washington, USA., 1993.

[2] C. Van Gysel, M. de Rijke, and E. Kanoulas. Neural Vector Spaces for Unsupervised Information Retrieval. *ACM Transactions on Information Systems (TOIS)*, 36(4):38:1–38:25, 2018.

[3] C.D. Manning, H. Schütze, and P. Raghavan. *Introduction to information retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[5] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS*, pages 3111–3119, 2013.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] B. Mitra and N. Craswell. An introduction to neural information retrieval. *Foundations and Trends in Information Retrieval*, 13(1):1–126, 2018.

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[10] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, B. Croft, and X. Cheng. A deep look into neural ranking models for information retrieval. *Information Processing & Management (IP&M)*, 57(6):102067, 2020.

[11] J. Lin, R. Nogueira, and A. Yates. Pretrained transformers for text ranking: Bert and beyond. *arXiv preprint arXiv:2010.06467*, 2020.

[12] K. D. Onal, Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. De Rijke, and M. Lease. Neural information retrieval: at the end of the early years. *Information Retrieval*, 21(2-3):111–182, 2018.

[13] C. Van Gysel, M. De Rijke, and E. Kanoulas. Neural vector spaces for unsupervised information retrieval. *ACM Transactions on Information Systems (TOIS)*, 36(4):38, 2018.

[14] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng. Text matching as image recognition. In *Proc. of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2793–2799, 2016.

[15] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. A deep relevance matching model for ad-hoc retrieval. In *Proc. of CIKM 2016*, pages 55–64, 2016.

[16] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Proc. of NIPS 2014*, pages 2042–2050, 2014.

[17] P. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. P. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proc. of CIKM 2013*, pages 2333–2338, 2013.

[18] X. Lu and H. Li. A deep architecture for matching short texts. In *Proc. of NIPS 2013*, pages 1367–1375, 2013.

[19] S. Bruch, S. Han, M. Bendersky, and M. Najork. A stochastic treatment of learning to rank scoring functions. In *Proc. of WSDM 2020*, pages 61–69, 2020.

[20] S. Bruch. An alternative cross entropy loss for learning-to-rank. *arXiv preprint arXiv:1911.09798*, 2019.

[21] S. Bruch, M. Zoghi, M. Bendersky, and M. Najork. Revisiting approximate metric optimization in the age of deep neural networks. In *Proc. of SIGIR 2019*, pages 1241–1244, 2019.

[22] C. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[23] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. *Proc. of NIPS 2006*, 19:193–200, 2006.

[24] S. Marchesin, A. Purpura, and G. Silvello. Focal elements of neural information retrieval models. an outlook through a reproducibility study. *Information Processing & Management (IP&M)*, page 102109, 2019.

[25] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[26] M. E Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[28] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. Cedr: Contextualized embeddings for document ranking. *arXiv preprint arXiv:1904.07094*, 2019.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. of NIPS 2017*, pages 5998–6008, 2017.

[30] E.M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing & Management (IP&M)*, 36(5):697–716, 2000.

[31] D. Ravana, Sri and A. Moffat. Score aggregation techniques in retrieval experimentation. In *Proc. of the Twentieth Australasian Conference on Australasian Database-Volume 92*, pages 57–66, 2009.

[32] M. Ferrante, N. Ferro, and L. Piazzon. s-AWARE: supervised measure-based methods for crowd-assessors combination. In *Proc. of CLEF 2020*, pages 16–27. Springer, 2020.

[33] H. Zhuang, X. Wang, M. Bendersky, and M. Najork. Feature transformation for neural ranking models. In *Proc. of SIGIR 2020*, 2020.

[34] P. Pobrotyn, T. Bartczak, M. Synowiec, R. Białobrzeski, and J. Bojar. Context-aware learning to rank with self-attention. *arXiv preprint arXiv:2005.10084*, 2020.

[35] L. Pang, Y. Lan, J. Guo, J. Xu, and X. Cheng. A study of matchpyramid models on ad-hoc retrieval. *arXiv preprint arXiv:1606.04648*, 2016.

[36] B. Muzellec and M. Cuturi. Generalizing point embeddings using the wasserstein space of elliptical distributions. In *NIPS 2018*, pages 10258–10269, 2018.

[37] C. Saedi, A. Branco, J. A. Rodrigues, and J. Silva. Wordnet embeddings. In *Proc. of The Third Workshop on Representation Learning for NLP: Rep4NLP@ACL*, pages 122–131, 2018.

[38] A. Purpura, K. Buchner, G. Silvello, and G.A. Susto. Neural feature selection for learning to rank. In *Proc. of ECIR 2021*. Springer, 2021.

[39] A. Purpura, M. Maggipinto, G. Silvello, and G.A. Susto. Probabilistic word embeddings in neural IR: A promising model that does not work as expected (for now). In *Proc. of ICTIR 2019*, pages 3–10, 2019.

[40] G. Salton. *Automatic information organization and retrieval*. McGraw-Hill, NY, USA, 1968.

[41] B. Croft, D. Metzler, and T. Strohman. *Search engines: Information retrieval in practice*, volume 520. Addison-Wesley Reading, 2010.

[42] H.P. Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

[43] K.S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[44] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proc. of SIGIR 1994*, pages 232–241, 1994.

[45] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. *SIGIR Forum*, 51(2):202–208, 2017.

[46] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proc. of SIGIR 1999*, 1999.

[47] D. Metzler and W.B. Croft. A markov random field model for term dependencies. In *Proc. of SIGIR 2005*, pages 472–479, 2005.

[48] G. Silvello, R. Bucco, G. Busato, G. Fornari, A. Langeli, A. Purpura, G. Rocco, A. Tezza, and M. Agosti. Statistical Stemmers: A Reproducibility Study. In Pasi et al. [199], pages 385–397.

[49] N. A. Jaleel, J. Allan, W. B. Croft, F. Diaz, L. S. Larkey, X. Li, M. D. Smucker, and C. Wade. Umass at TREC 2004: Novelty and HARD. In *TREC 2004*, 2004.

[50] M. Efron, P. Organisciak, and K. Fenlon. Improving retrieval of short texts through document expansion. In *Proc. of SIGIR 2012*, pages 911–920, 2012.

[51] K.D. Onal, Y. Zhang, I.S. Altingovde, M.M. Rahman, P. Karagoz, A. Braylan, B. Dang, H.L. Chang, H. Kim, Q. Mcnamara, A. Angert, E. Banner, V. Khetan, T. Mcdonnell, A.T. Nguyen, D. Xu, B.C. Wallace, M. Rijke, and M. Lease. Neural information retrieval: At the end of the early years. *Information Retrieval*, 21(2–3):111–182, June 2018.

[52] Z. A. Yilmaz, S. Wang, W. Yang, H. Zhang, and J. Lin. Applying bert to document retrieval with birch. In *Proc. of EMNLP-IJCNLP 2019*, pages 19–24, 2019.

[53] Z. Dai and J. Callan. Deeper text understanding for ir with contextual neural language modeling. In *Proc. of SIGIR 2019*, pages 985–988, 2019.

[54] V. Karpukhin, B. Oğuz, S. Min, L. Wu, S. Edunov, D. Chen, and W.T. Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[55] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[56] N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7(3):183–204, 1989.

[57] M. Wong, Y. Cai, and Y. Yao. Computation of term associations by a neural network. In *Proc. of SIGIR 1993*, pages 107–115, 1993.

[58] T. Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011.

[59] N. Fuhr. Some Common Mistakes In IR Evaluation, And How They Can Be Avoided. *SIGIR Forum*, to appear, 2017.

[60] C. Quoc and V. Le. Learning to rank with nonsmooth cost functions. *Proc. of NIPS 2007*, 19:193–200, 2007.

[61] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. of ICLR 2005*, pages 89–96, 2005.

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

[63] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[64] T. Qin, T. Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374, 2010.

[65] C. Cleverdon. The cranfield tests on index language devices. In *Aslib proceedings*. MCB UP Ltd, 1967.

[66] E.M. Voorhees. The philosophy of information retrieval evaluation. In *Proc. of CLEF 2001*, pages 355–370. Springer, 2001.

[67] E. M. Voorhees. The TREC Robust Retrieval Track. *ACM SIGIR Forum*, 39(1):11–20, 2005.

[68] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[69] J. Kekäläinen and J. Järvelin. Using graded relevance assessments in ir evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002.

[70] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proc. of CIKM 2009*, pages 621–630, 2009.

[71] G. M. Di Nunzio, N. Ferro, T. Mandl, and C. Peters. CLEF 2006: Ad Hoc Track Overview. In C. Peters, P. Clough, F. C. Gey, J. Karlgren, B. Magnini, D. W. Oard, M. de Rijke, and M. Stempfhuber, editors, *Proc. of CLEF 2006*, pages 21–34. Lecture Notes in Computer Science (LNCS) 4730, Springer, Heidelberg, Germany, 2007.

[72] E. Agirre, G. M. Di Nunzio, N. Ferro, T. Mandl, and C. Peters. CLEF 2008: Ad Hoc Track Overview. In C. Peters, T. Deselaers, N. Ferro, J. Gonzalo, G. J. F. Jones, M. Kurimo, T. Mandl, and A. Peñas, editors, *Proc. of CLEF 2008*, pages 15–37. Lecture Notes in Computer Science (LNCS) 5706, Springer, Heidelberg, Germany, 2009.

[73] W. R. Hersh, C. Buckley, T. J. Leone, and D. Hickam. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proc. of SIGIR 1994*, pages 192–201. Springer, 1994.

[74] D. Harman. The darpa tipster project. In *ACM SIGIR Forum*, volume 26, pages 26–28. ACM, 1992.

[75] D. Harman. Document detection data preparation. In *TIPSTER TEXT PROGRAM: PHASE I: Proc. of a Workshop held at Fredricksburg, Virginia, September 19-23, 1993*, 1993.

[76] D. Hawking, E. M. Voorhees, N. Craswell, and P. Bailey. Overview of the TREC-8 Web Track. In E. M. Voorhees and D. K. Harman, editors, *Proc. of TREC-8*. National Institute of Standards and Technology (NIST), Special Publication 500-246, Washington, USA., 1999.

[77] C. Clarke, N. Craswell, and I. Soboroff. Overview of the trec 2009 web track. Technical report, Waterloo University (Ontario), 2009.

[78] T. Qin and T. Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

[79] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[80] J. Allan, B. Carterette, J. A Aslam, V. Pavlu, B. Dachev, and E. Kanoulas. Million query track 2007 overview. Technical report, TREC-NIST, 2007.

[81] H. Schütze, C.D. Manning, and P. Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, Cambridge, UK, 2008.

[82] N. Craswell, W. B. Croft, J. Guo, B. Mitra, and M. de Rijke. Neu-IR: The SIGIR 2016 Workshop on Neural Information Retrieval. In *Proc. of SIGIR 2016*, pages 1245–1246. ACM Press, New York, USA, 2016.

[83] N. Craswell, W. B. Croft, M. de Rijke, J. Guo, and B. Mitra. SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR). In *Proc. of SIGIR 2017*, pages 1431–1432. ACM Press, New York, USA, 2017.

[84] B. Mitra and N. Craswell. An Introduction to Neural Information Retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, December 2018.

[85] N. Craswell, W. B. Croft, M. de Rijke, J. Guo, and B. Mitra. Neural information retrieval: introduction to the special issue. *Information Retrieval*, 21(2-3):107–110, 2018.

[86] M. Agosti, E. Fabris, and G. Silvello. On Synergies Between Information Retrieval and Digital Libraries. In *Proc of IRCDL 2019*, volume 988 of *Communications in Computer and Information Science*, pages 3–17. Springer, 2019.

[87] J. Lin. The neural hype and comparisons against weak baselines. *SIGIR Forum*, 52(2):40–51, January 2019.

[88] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A Neural Click Model for Web Search. In Bourdeau et al. [200], pages 531–541.

[89] B. Mitra, F. Diaz, and N. Craswell. Learning to match using local and distributed representations of text for web search. In *Proc. of WWW 2017*, pages 1291–1299, 2017.

[90] H. Zamani, B. Mitra, X. Song, N. Craswell, and S. Tiwary. Neural ranking models with multiple document fields. In *Proc. of WSDM 2018*, pages 700–708, 2018.

[91] G. Zuccon, B. Koopman, P. Bruza, and L. Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In *Proc. of ADCS 2015*, pages 1–8, 2015.

[92] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 55–64, 2016.

[93] L. Boytsov, D. Novak, Y. Malkov, and E. Nyberg. Off the beaten path: Let's replace term-based retrieval with k-nn search. In *Proc. of CIKM 2016*, pages 1099–1108, 2016.

[94] S. Yelong, H. Xiaodong, G. Jianfeng, D. Li, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proc. of CIKM 2014*, pages 101–110. ACM Press, New York, USA, 2014.

[95] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward. Semantic modelling with long-short-term memory for information retrieval. *arXiv preprint arXiv:1412.6629*, 2014.

[96] Q. Ai, L. Yang, J. Guo, and W. B. Croft. Improving language estimation with the paragraph vector model for ad-hoc retrieval. In *Proc. of SIGIR 2016*, pages 869–872. ACM Press, New York, USA, 2016.

[97] D. Ganguly, D. Roy, M. Mitra, and G. J. F. Jones. Word embedding based generalized language model for information retrieval. In *Proc. of SIGIR 2015*, pages 795–798. ACM Press, New York, USA, 2015.

[98] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. Semantic matching by non-linear word transportation for information retrieval. In *Proc. of CIKM 2016*, pages 701–710. ACM Press, New York, USA, 2016.

[99] I. Vulić and M. F. Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proc. of SIGIR 2015*, pages 363–372. ACM, ACM Press, New York, USA, 2015.

[100] C. Van Gysel, M. de Rijke, and E. Kanoulas. Learning latent vector spaces for product search. In Mukhopadhyay et al. [201], pages 165–174.

[101] C. Van Gysel, M. de Rijke, and M. Worring. Unsupervised, efficient and semantic expertise retrieval. In Bourdeau et al. [200], pages 1069–1079.

[102] C. Van Gysel, M. de Rijke, and E. Kanoulas. Mix'n Match: Integrating Text Matching and Product Substitutability within Product Search. In *Proc. CIKM 2018*, pages 1373–1382. ACM Press, New York, USA, 2018.

[103] S. Wan, Y. Lan, J. Xu, J. Guo, L. Pang, and X. Cheng. Match-srnn: Modeling the recursive matching structure with spatial RNN. *arXiv preprint arXiv:1604.04378*, 2016.

[104] T. Fan, J. Guo, Y. Lan, J. Xu, C. Zhai, and X. Cheng. Modeling diverse relevance patterns in ad-hoc retrieval. In *Proc. of SIGIR 2018*, pages 375–384. ACM Press, New York, USA, 2018.

[105] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proc. of SIGIR 2017*, pages 55–64. ACM Press, New York, USA, 2017.

[106] Z. Dai, C. Xiong, J. Callan, and Z. Liu. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *Proc. of WSDM 2018*, pages 126–134. ACM Press, 2018.

[107] K. Hui, A. Yates, K. Berberich, and G. de Melo. Pacrr: A position-aware neural ir model for relevance matching. *arXiv preprint arXiv:1704.03940*, 2017.

[108] K. Hui, A. Yates, K. Berberich, and G. de Melo. Co-pacrr: A context-aware neural ir model for ad-hoc retrieval. In *Proc. WSDM 2018*, WSDM '18, pages 279–287, New York, NY, USA, 2018. ACM.

[109] J. Wei, L. Kuang, Peilin Y., and J. Lin. Critically examining the" neural hype": Weak baselines and the additivity of effectiveness gains from neural ranking models. In *Proc. of SIGIR 2019*, pages 1129–1132. ACM Press, 2019.

[110] N. Ferro, N. Fuhr, K. Järvelin, N. Kando, M. Lippold, and J. Zobel. Increasing Reproducibility in IR: Findings from the Dagstuhl Seminar on "Reproducibility of Data-Oriented Experiments in e-Science". *SIGIR Forum*, 50(1):68–82, 2016.

[111] J. Arguello, M. Crane, F. Diaz, J. Lin, and A. Trotman. Report on the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum*, 49(2):107–116, December 2015.

[112] R. Clancy, N. Ferro, C. Hauff, J. Lin, T. Sakai, and Z. Z. Wu. The SIGIR 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In *Proc. of SIGIR 2019*, pages 1432–1434. ACM Press, 2019.

[113] Norbert Fuhr. The PRIMAD Model of Reproducibility. Dagstuhl Seminar 16111 - Rethinking Experimental Methods in Computing, March 2016.

[114] N. Ferro, N. Fuhr, and A. Rauber. Introduction to the Special Issue on Reproducibility in Information Retrieval: Evaluation Campaigns, Collections, and Analyses. *J. Data and Information Quality*, 10(3):9:1–9:4, 2018.

[115] N. Ferro, N. Fuhr, and A. Rauber. Introduction to the special issue on reproducibility in information retrieval: Tools and infrastructures. *J. Data and Information Quality*, 10(4):14:1–14:4, 2018.

[116] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. MacDonald, and S. Vigna. Toward reproducible baselines: The open-source IR reproducibility challenge. In *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proc.*, pages 408–420. Lecture Notes in Computer Science (LNCS) 9626, Springer, Heidelberg, Germany, 2016.

[117] P. Yang, H. Fang, and J. Lin. Anserini: Reproducible ranking baselines using lucene. *J. Data and Information Quality*, 10(4):16:1–16:20, 2018.

[118] N. Ferro and G. Silvello. Rank-Biased Precision Reloaded: Reproducibility and Generalization. In *Proc. of ECIR 2015*, pages 768–780. Lecture Notes in Computer Science (LNCS) 9022, Springer, Heidelberg, Germany, 2015.

[119] F. Hopfgartner, A. Hanbury, H. Müller, I. Eggel, K. Balog, t. Brodt, G. V. Cormack, J. Lin, J. Kalpathy-Cramer, N. Kando, M. P. Kato, A. Krithara, T. Gollub, M. Potthast, E. Viegas, and S. Mercer. Evaluation-as-a-Service for the Computational Sciences: Overview and Outlook. *Journal of Data and Information Quality*, 10(4):15:1–15:32, 2018.

[120] F. Hasibi, K. Balog, and S. E. Bratsberg. Exploiting Entity Linking in Queries for Entity Retrieval. In *Proc. of ICTIR 2016*, pages 209–218. ACM Press, New York, USA, 2016.

[121] A. Dür, A. Rauber, and P. Filzmoser. Reproducing a Neural Question Answering Architecture Applied to the SQuAD Benchmark Dataset: Challenges and Lessons Learned. In Pasi et al. [199], pages 102–113.

[122] X. Yang, I. Ounis, R. McCreadie, C. Macdonald, and A. Fang. On the Reproducibility and Generalisation of the Linear Transformation of Word Embeddings. In Pasi et al. [199], pages 263–275.

[123] N. Ferro, S. Marchesin, A. Purpura, and G. Silvello. A Docker-Based Replicability Study of a Neural Information Retrieval Model. In *Proc. of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019.*, pages 37–43, 2019.

[124] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)*, 20(4):357–389, 2002.

[125] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[126] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[127] W. Yang, H. Zhang, and J. Lin. Simple applications of BERT for ad hoc document retrieval. *arXiv preprint arXiv:1903.10972*, 2019.

[128] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proc. of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[129] J. P. Callan, W. B. Croft, and J. Broglio. Trec and tipster experiments with inquery. *Information Processing & Management (IP&M)*, 31(3):327–343, 1995.

[130] R. Krovetz. Viewing Morphology as an Inference Process. In R. Korfhage, E. Rasmussen, and P. Willett, editors, *Proc. of SIGIR 1993*, pages 191–202. ACM Press, New York, USA, 1993.

[131] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall/CRC, USA, 1995.

[132] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[133] C. Macdonald, R. McCreadie, R.L. Santos, and I. Ounis. From puppy to maturity: Experiences in developing terrier. *Proc. of OSIR at SIGIR*, pages 60–63, 2012.

[134] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[135] S. Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.

[136] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[137] C. Van Gysel, E. Kanoulas, and M. de Rijke. Pyndri: a Python interface to the indri search engine. In *Proc. ECIR 2018*, pages 744–748. Springer, Springer, Heidelberg, Germany, 2017.

[138] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the 13th international conference on artificial intelligence and statistics (AISTATS)*, pages 249–256, 2010.

[139] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. In *Proc. of the International Conference on Intelligent Analysis*, volume 2, pages 2–6. Citeseer, 2005.

[140] E. M. Voorhees. Overview of the TREC 2004 robust track. In *TREC 2004*, 2004.

[141] J. Allan, D. Harman, E. Kanoulas, D. Li, C. Van Gysel, and E. Voorhees. TREC 2017 common core track overview. In *Proc. of The Twenty-Sixth Text REtrieval Conference, TREC 2017*. National Institute of Standards and Technology (NIST), Special Publication 500-324, Washington, USA., 2017.

[142] C. Van Gysel, D. Li, and E. Kanoulas. ILPS at TREC 2017 Common Core Track. *arXiv preprint arXiv:1801.10603*, 2018.

[143] N. Ferro and C. Peters. CLEF 2009 Ad Hoc Track Overview: TEL & Persian Tasks. In F. Borri, A. Nardi, and C. Peters, editors, *Working Notes for the CLEF 2009 Workshop*. Published Online, 2009.

[144] K. P. Burnham and D. R. Anderson. *Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach*. Springer-Verlag, Heidelberg, Germany, 2nd edition, 2002.

[145] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS 2013*, pages 3111–3119. Curran Associates, Inc., 2013.

[146] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[147] H. Zamani and W. B. Croft. Relevance-based word embedding. In *Proc. of SIGIR 2017*, pages 505–514, 2017.

[148] H. Zamani, M. Dehghani, W. B. Croft, E. G. Learned-Miller, and J. Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *CIKM 2018*, pages 497–506, 2018.

[149] L. Vilnis and A. McCallum. Word representations via gaussian embedding. In *ICLR*, 2015.

[150] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov. Learning word vectors for 157 languages. In *Proc. of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[151] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019.

[152] C. Fellbaum. Wordnet: An electronic lexical database. *Computational Linguistics*, 25(2):292–296, June 1999.

[153] J. B. Gao, B. W. Zhang, and X. H. Chen. A wordnet-based semantic similarity measurement combining edge-counting and information content theory. *Engineering Applications of Artificial Intelligence*, 39:80–88, 2015.

[154] V. M. Ngo, T. H. Cao, and T. M. V. Le. Wordnet-based information retrieval using common hypernyms and combined features. *arXiv preprint arXiv:1807.05574*, 2018.

[155] G. Varelas, E. Voutsakis, P. Raftopoulou, E. G. M. Petrakis, and E. E. Milios. Semantic similarity methods in wordnet and their application to information retrieval on the web. In *Proc. of the 7th Annual ACM International Workshop on Web Information and Data Management*, WIDM '05, pages 10–16, New York, NY, USA, 2005. ACM.

[156] C. Basu, L. Dietz, and C. Fellbaum. Wordnetcontext: Information retrieval-friendly access to wordnet senses. In *Proc. of S/KG4IR/Data:Search@SIGIR*, volume 2127 of *CEUR Workshop Proceedings*, pages 63–64. CEUR-WS.org, 2018.

[157] E. M. Voorhees. Query expansion using lexical-semantic relations. In *Proc. of SIGIR 1994*, pages 61–69, 1994.

[158] S. Liu, F. Liu, C. T. Yu, and W. Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *Proc. of SIGIR 2004*, pages 266–272, 2004.

[159] R. Mihalcea and D. Moldovan. Semantic indexing using wordnet senses. In *Proc. of the ACL-2000 Workshop on Recent Advances in Natural Language Processing and Information Retrieval*, volume 11 of *RANLPIR '00*, pages 35–45, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[160] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proc. of ACL 1994*, ACL '94, pages 133–138, Stroudsburg, PA, USA, 1994. ACL.

[161] Rothe S. and H. Schütze. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proc. of ACL 2015*, volume 1, pages 1793–1803, 2015.

[162] X. Li, L. Vilnis, D. Zhang, M. Boratko, and A. McCallum. Smoothing the geometry of probabilistic box embeddings. In *Proc. of ICLR 2019*, 2019.

[163] F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[164] M. Hosseini, I. Cox, N. Milić-Frayling, G. Kazai, and V. Vinay. On aggregating labels from multiple crowd workers to infer relevance of documents. In *Proc. of ECIR 2012*, pages 182–194. Springer, 2012.

[165] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information processing & management (IP&M)*, 51(6):757–772, 2015.

[166] Zhen Q., Le Y., Honglei Z., Yi T., Rama K.P., Xuanhui W., Michael B., and Marc N. Neural rankers are hitherto outperformed by gradient boosted decision trees. In *Proc. of ICLR*, 2021.

[167] R.K. Pasumarthi, H. Zhuang, X. Wang, M. Bendersky, and M. Najork. Permutation equivariant document interaction network for neural learning to rank. In *Proc. of ICTIR 2020*, pages 145–148, 2020.

[168] W. Chen, T. Liu, Y. Lan, Z. Ma, and H. Li. Ranking measures and loss functions in learning to rank. *Proc. of NIPS 2009*, 22:315–323, 2009.

[169] X. Liu, J. Van De Weijer, and A.D. Bagdanov. Leveraging unlabeled data for crowd counting by learning to rank. In *Proc. of CVPR 2018*, pages 7661–7669, 2018.

[170] T. Qin, T. Liu, and H. Li. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, 13(4):375–397, 2010.

[171] H. Zamani and W.B. Croft. On the theory of weak supervision for information retrieval. In *Proc. of ICTIR 2018*, pages 147–154, 2018.

[172] H. Zhuang, X. Wang, M. Bendersky, A. Grushetsky, Y. Wu, P. Mitrichev, E. Sterling, N. Bell, W. Ravina, and H. Qian. Interpretable learning-to-rank with generalized additive models. *arXiv preprint arXiv:2005.02553*, 2020.

[173] O. Alonso. The practice of crowdsourcing. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 11(1):1–149, 2019.

[174] M. Lease and G. Kazai. Overview of the trec 2011 crowdsourcing track. In *Proc. of TREC 2011*, 2011.

[175] M. Smucker, G. Kazai, and M. Lease. Overview of the trec 2012 crowdsourcing track. In *Proc. of TREC 2012*, 2012.

[176] M Smucker, G. Kazai, and M. Lease. Overview of the trec 2013 crowdsourcing track. In *Proc. of TREC 2013*, 2013.

[177] C.M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[178] R. Agrawal. Finite-sample concentration of the multinomial in relative entropy. *IEEE Transactions on Information Theory*, 66(10):6297–6302, 2020.

[179] S. Sun and K. Duh. Modeling document interactions for learning to rank with regularized self-attention. *arXiv preprint arXiv:2005.03932*, 2020.

[180] M. Ibrahim and M. Carman. Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank. *ACM Transactions on Information Systems (TOIS)*, 34(4):1–38, 2016.

[181] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proc. of NIPS 2017*, pages 3146–3154, 2017.

[182] M. Ferrante, N. Ferro, and E. Losiouk. How do interval scales help us with better understanding IR evaluation measures? *Information Retrieval*, 23(3):289–317, 2020.

[183] N. Fuhr. Some Common Mistakes In IR Evaluation, And How They Can Be Avoided. *SIGIR Forum*, 51, 2018.

[184] M. Köppel, A. Segner, M. Wagener, L. Pensel, A. Karwath, and S. Kramer. Pairwise learning to rank by neural networks revisited: Reconstruction, theoretical analysis and practical performance. In *Proc. of KDD 2019*, pages 237–252. Springer, 2019.

[185] F. Xia, T. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proc. of ICML 2008*, pages 1192–1199, 2008.

[186] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proc. of CIKM 2017*, pages 257–266, 2017.

[187] Q. Ai, K. Bi, J. Guo, and W.B. Croft. Learning a deep listwise context model for ranking refinement. In *Proc. of SIGIR 2018*, pages 135–144, 2018.

[188] A. Gigli, C. Lucchese, F.M. Nardini, and R. Perego. Fast feature selection for learning to rank. In *Proc. of ICTIR 2016*, pages 167–170, 2016.

[189] X. Geng, T. Liu, T. Qin, and H. Li. Feature selection for ranking. In *Proc. of SIGIR 2007*, pages 407–414, 2007.

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

[191] J. C Gower and G. Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.

[192] A. Rahangdale and S. Raut. Deep neural network regularization for feature selection in learning-to-rank. *IEEE Access*, 7:53988–54006, 2019.

[193] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. In *Proc. of NIPS 2018*, pages 9505–9515, 2018.

[194] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[195] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[196] A. Tonon and F. Vandin. Permutation strategies for mining significant sequential patterns. In *Proc. of ICDM 2019*, pages 1330–1335. IEEE, 2019.

[197] T. Qin and T.Y. Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

[198] A. Rutherford. *ANOVA and ANCOVA. A GLM Approach*. John Wiley & Sons, New York, USA, 2nd edition, 2011.

[199] G. Pasi, B. Piwowarski, L. Azzopardi, and A. Hanbury, editors. *Proc. of the 40th European Conference on IR Research, ECIR 2018*, volume 10772 of *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 2018.

[200] Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors. *Proc. of the 25th International Conference on World Wide Web, WWW 2016*. ACM, 2016.

[201] Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors. *Proc. of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016*. ACM Press, New York, USA, 2016.