Study on Feature Interactions in Multiple Field Document Ranking

Kentaro Takiguchi

Department of Engineering, University of Bristol

December 25, 2020

Abstract

We are surrounded by various search applications that can help us in many situations, such as finding knowledge from the internet, locating the best place to stay for the next trip, discovering recipes that can be prepared using leftovers in the fridge, etc.

Neural information retrieval has received much attention in recent years because of its ability to capture more complex interactions than conventional machine learning models. However, whereas numerous neural ranking models have been proposed for single-field document ranking, not much progress has been made on semi-structured multi-field document ranking, although the quality of search directly impacts revenue for companies.

In this thesis, I discuss document ranking models for modern search applications from the relationship between the ranking task and the recommendation task. I hypothesized that the multi-field document ranking task is located somewhere between single-field document ranking and recommendation. If this hypothesis is correct, we can employ techniques to capture text matching signals from information retrieval and techniques to combine multiple evidence from recommender systems. The key contributions of this project are as follows:

- I reviewed the entire history of information retrieval from an academic and industrial perspective, and discussed the similarity between document ranking and item recommendation.
- My experiments has shown the possibility that recommendation models work for document ranking as well. It has also shown that incorporating the characteristics of document ranking into recommendation models further improves performance.
- I have published the implementation on GitHub for the validity of my experiments.

Acknowledgements

I would like to thank my supervisor, Dr. Luis Vaquero Gonzalez, for his constant guidance throughout this project. His feedback on early drafts of this paper helped to clarify the direction of my research. I am extremely grateful to him for his patience in helping me realize my abstract ideas. I would also like to thank Cookpad and the researchers for providing the dataset for this project and discussing the implementation.

Contents

1 Introduction **6**

1.1 Aims and objectives 8

2 Background and Context **11**

2.1 Notation 11

2.2 Problem definition 12

2.3 Utilities and metrics 13

2.3.1 Set-based measures 13

2.3.2 Rank-based measures 14

2.4 Probabilistic retrieval 16

2.5 Vector space model 19

2.6 Limitation of traditional methods 19

3 Conventional Learning to Rank **21**

3.1 Pointwise approach 22

3.2 Pairwise approach 23

3.2.1 Pairwise hinge loss 23

3.2.2 Pairwise cross-entropy loss 23

3.3 Listwise approach 24

4 Neural Information Retrieval **25**

4.1 Text Representation 25

4.2 Types of ranking models 27

4.2.1 Representation-based models 27

4.2.2 Interaction-based models 29

4.2.3 Hybrid models . . . 29
4.3 Matching with multiple document fields . . . 30
4.4 Information retrieval and recommendation . . . 31
4.4.1 Factorization machines . . . 32
4.5 Examples of industrial application . . . 34
4.6 Categorizing search applications . . . 35
4.6.1 Keyword matching vs. free text matching . . . 35
4.6.2 Long text vs. short text . . . 36
4.6.3 General vs. domain specific . . . 36
4.6.4 Ranking vs. re-ranking . . . 37
4.6.5 Web search and other industrial search applications . . . 37
4.7 Validity of experiments on neural models . . . 38

5 Neural Ranking Models for Multiple Field Documents 40
5.1 Dataset . . . 41
5.1.1 Recipe search dataset . . . 41
5.2 Data processing and modeling . . . 43
5.3 Training and validation . . . 44
5.4 RQ1. Is it important to learn feature interactions? . . . 45
5.4.1 Experimental setup . . . 45
5.4.2 Results and Discussion . . . 46
5.5 RQ2. Is it necessary to limit feature interactions? . . . 47
5.5.1 Experimental setup . . . 47
5.5.2 Results and Discussion . . . 48
5.6 RQ3. Does feeding first-order features contribute to effectiveness? . . . 49
5.6.1 Experimental setup . . . 49
5.6.2 Result and Discussion . . . 49
5.7 RQ4. Are there any other important feature interactions besides query-fields? . . . 50
5.7.1 Experimental setup . . . 51
5.7.2 Results and discussion . . . 51

6 Conclusion **54**
6.1 Summary of outcomes 54
6.2 Summary of contributions 54
6.3 Areas for improvement and future work 55

Chapter 1

Introduction

Search is now an essential tool for everyone to locate the content suited for our needs from the overwhelming amount of information for better decision making. We use a variety of search applications daily. The quality of such search applications is crucial not only for users but also for companies because it directly impacts revenue. Many companies have been making great efforts to improve the performance of their search systems.

The information retrieval community has mostly focused on simple document retrieval, in which given a query, estimate which text is more relevant. This experimental setting is still mainstream in today's literature. However, with the development of the internet, search applications are becoming more complex to address growing diverse demands. Figure 1.1 shows example documents from industrial search applications (Amazon [1], Airbnb [2], and Cookpad [3]).

Documents in modern search applications differ from those employed in classical task settings in many ways. For example, documents in modern search applications usually consist of various fields such as title, description, image, review score, facilities, etc. In such applications, traditional methods may fall short in many cases, such as capturing text significance from short text fields, encoding non-text fields, and combining different fields' scores. It has resulted that traditional methods based on the assumptions made in the 1980s may not be practical any longer.

[1] `https://amazon.com`
[2] `https://airbnb.com`
[3] `https://cookpad.com`

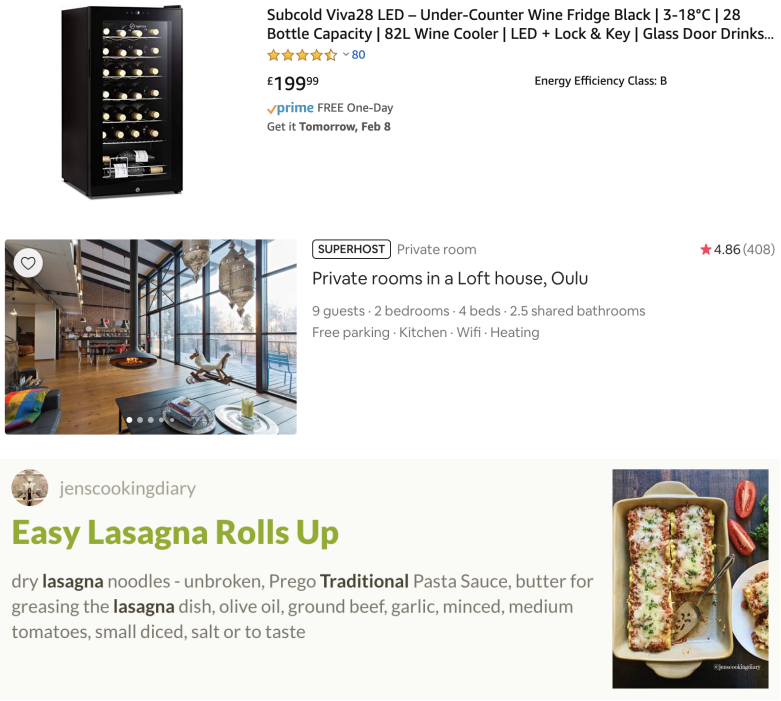


Figure 1.1: Examples of industrial search applications (Amazon, Airbnb, Cookpad)

Consequently, practical search applications often employ heuristic methods to supplement information (synonyms, non-text fields, feature interactions, etc.) that traditional scoring methods do not consider. The heuristic parameter tuning is often suboptimal. The system would suffer from endless empirical performance tuning because of the vast number of parameters and the system's dynamic nature, where the meaning of terms and user behavior change day by day.

Neural information retrieval is recently receiving increased attention. The percentage of neural information retrieval-related papers published at SIGIR has risen dramatically from 1% in 2014 to 79% by 2020. That is because neural models are thought to be better suited for incorporating natural language properties, such as synonymy and ambiguity.

Neural ranking models can be classified into two types: representation-based models and interaction-based models. Representation-based models separately learn a representation for query and document and then compute the relevance score using a matching function. On the other hand, interaction-based models learn representations by inter-

acting query and document from the first layer. Interaction-based models are said to be superior in performance than representation-based models because representation-based models construct each representation without knowing the structure of the other side. Many variants have been proposed in both architecture to better model relevance between query and document, but they were mainly designed for single-field document ranking.

To the best of my knowledge, NRM-F proposed by Zamani et al. [78] is the only paper that discusses how neural models can deal with multiple document fields from an architectural perspective. The authors say that it is better for the ranker to score the whole document jointly, rather than generate a per-field score and aggregate. It outperformed a learning to rank baseline with hand-crafted features, but some questions were raised.

When designing a ranking model, we have to make several architecture decisions, such as representation-based vs. interaction-based, which feature interactions to learn, how to aggregate scores, etc. NRM-F can be categorized into an interaction-based model that learns query-field interactions and concatenates feature vectors to aggregate scores, but how do the unselected options affect effectiveness? Multiple field document ranking is still in its infancy, and many things are not known yet.

Whereas multiple-field document ranking is a relatively minor topic in information retrieval, how to model semi-structured data has been a central topic in the field of recommender systems. Those two fields are known to share some properties as shown in Table 1.2, I hypothesized that the ranking task in modern search application is located somewhere between traditional document ranking (single-field document ranking) and item recommendation. Based on this idea, I examine different architectures, mainly focusing on interaction learning between fields, and demystify how ranking models work while targeting practical use cases.

1.1 Aims and objectives

The overall aim of this project is to provide insights for designing a ranking model for semi-structured documents. In particular, I focus on feature interactions and address the following research questions through experiments.

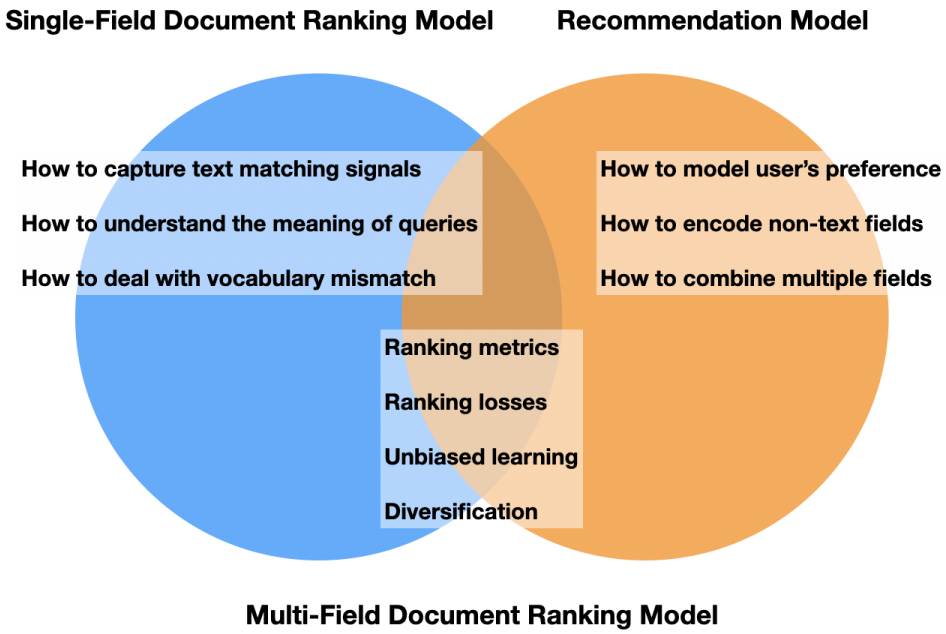


Figure 1.2: Venn diagram showing how each field overlaps

RQ1. Is it important to learn feature interactions?

In the first place, is learning interactions important? I compare the performance of representation-based and interaction-based models. I then measure the effectiveness of limiting feature interactions.

RQ2. Is it necessary to limit feature interactions?

NRM-F explicitly learns query-field interactions, whereas recommendation models usually do not distinguish between a query and item features. I do the opposite of what the literature says and comparing their performance.

RQ3. Does feeding first-order features contribute to effectiveness?

NRM-F does not consider first-order features, but they may have essential signals that improve effectiveness. I compare models with and without first-order features.

RQ4. Are there any other important feature interactions besides query-fields?

While NRM-F focuses on query-field interactions, I assume that there must be other important feature interactions to consider. I measure and visualize feature interactions to locate such interactions.

The rest of the thesis is organized as follows: In chapter 2, I walk you through the brief history of information retrieval that gives the context to understand the challenges I am tackling. In chapter 3, I describe the fundamentals for better understanding of ranking models. In chapter 4, I discuss various recent neural ranking models in detail. In chapter 5, I describe the experimental design and discuss the results. Finally, in chapter 6, I conclude my work and propose directions for future research on the topic.

Chapter 2

Background and Context

The first commercial information retrieval system was found in the 1960s, even before the emergence of the internet. Since then, researchers have attempted to improve the performance of search. The most classical style of information retrieval was the so-called Boolean retrieval. A query was represented as a logical combination of terms, and the information retrieval system returns a set of the documents that exactly matched a given query. Since this approach did not sort retrieved documents, methods that can rank documents in some rational way began to be explored with the rise of the demand for retrieving a large number of documents, where users cannot assess the relevance of all the retrieved documents one by one. This chapter describes how researchers have tackled document ranking in the long history of information retrieval. I first introduce the notation used in this thesis. Next, I define what document ranking is and how to compare the performance of methods. Then, I describe two classic but still major approaches for ranked retrieval.

2.1 Notation

I adopt some common notation for this thesis shown in 2.1. In this thesis, I use terms *documents* and *items* interchangeably.

Notation	Meaning
q	Singe query
d	Single document
Q	Set of queries
D	Set of documents
t_q	Term in query q
t_d	Term in document d
$s_{q,d}$	Relevance score against a pair of query q and document d
R_q	Set of ranked results retrieved for query q
$rel(q, d)$	Degree of relevance of document d to query q
$rel(q, d_i) > rel(q, d_j)$	d_i is more relevant than d_j for query q
$TF(t, d)$	Frequency of term t in document d
$DF(t)$	Number of documents that contain term t
$IDF(t)$	Inverse of DF
$\mathbf{v}_z$	Vector representation of text z
F	Scoring function
Φ	Mapping function
Λ	Aggregation function

Table 2.1: Notation used in this thesis

2.2 Problem definition

The general definition of the task is to make a function $f(q, D)$, where q is a query, and D is a set of documents, to produce a permutation of items that maximizes the utility of the list of retrieved documents.

If the system ranks documents directly from the entire collection, it is classified as a ranking task. If the system ranks documents against a set of generated candidates for which irrelevant documents have already been filtered out, it is classified as a re-ranking task. The latter system is specifically referred to as a multi-stage system. The ranking task differs from the re-ranking task in two ways: size and relevance within a set. Multi-stage search systems are usually adopted in industry because of their rigorous performance requirements.

2.3 Utilities and metrics

We have seen that an information retrieval system produces a permutation of items. How do we know which algorithm is more effective? Evaluation is an issue that has been discussed for a long time in the information retrieval community [65], but why does it matter?

Ultimately, the system's goal is to maximize business metrics such as revenue and the number of active users. Still, those metrics are not possible to directly maximize because it involves various factors, and what is critical for the system is difficult to know. For that reason, search systems usually have user-based metrics such as Click-Through Rate, Session Success Rate, Session Abandonment Rate, Zero Result Rate, etc., and metrics are selected based on their correlation with business metrics. User-based metrics are also called online metrics because they are measured based on actual user behavior. On the one hand, online metrics can represent how good the search system is for users from different aspects, but on the other hand, it is not suitable to iteratively improve the ranking algorithm itself because it involves the process of collecting logs in a production environment. Researchers typically use offline metrics to make comparisons between the ranked lists output by the model. Let us see several offline metrics in the following subsections.

2.3.1 Set-based measures

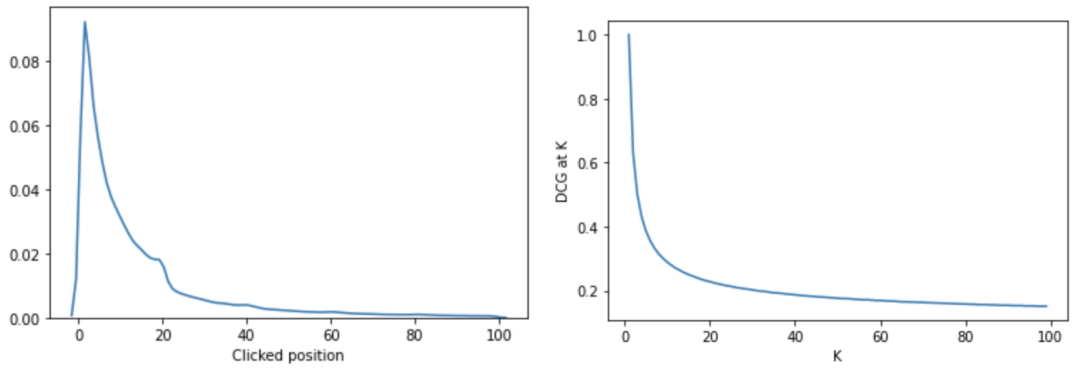
Recall, Precision, and F1-score are called set-based metrics because they are order-insensitive. Precision was proposed in 1966 as part of the Cranfield Project. The basic idea is that the precision of a retrieval system for a certain query is the proportion of relevant results defined as follows.

$$\text{Precision} = \frac{\text{relevant retrieved results}}{\text{overall retrieved results}}$$

Similarly, Recall represents how much relevant documents are actually retrieved defined as follows.

$$\text{Recall} = \frac{\text{relevant retrieved results}}{\text{relevant results in database}}$$

F1-score combines these two aspects of retrieval.



(a) The distribution of clicked positions

(b) DCG at position k

Figure 2.1: The distribution of clicked positions and gain

$$\mathrm{F}_1 = 2\frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

Recall and Precision measures are generally a trade-off, meaning that if a search system returns more results, Recall can only increase, but Precision is likely to fall.

These metrics treat all retrieved results equally, but as the user's attention decreases, documents at lower positions may not be examined and less critical. In a sense, ranked-based measures described below are more suitable to evaluate ranking models.

2.3.2 Rank-based measures

Top-heaviness is a central property in information retrieval. This is because the higher a system ranks relevant documents, the more likely it is that users will be able to achieve their goals in less time. Mean Average Precision (MAP) is one of the most common metric that has the top-heaviness property. MAP considers the rank position of each relevant document. Then, the precision is averaged by dividing the sum of discounted precisions by the total number of relevant documents defined as follows.

$$\mathrm{MAP@k} = \frac{1}{|Q|}\sum_{q}\frac{1}{k}\left(\sum_{i}^{k}\mathrm{rel}(q,d_i)\frac{\sum_j^i \mathrm{rel}(q,d_j)}{i}\right)$$

where Q is a set of queries examined, k is the cut-off position, d_i is the i^{th}-ranked document returned by the system, and rel is the binary relevance assessment assigning 1

to relevant and 0 to non-relevant results. Though MAP has been used in many experiments, it is said to be based on unrealistic assumptions about user behaviour. Some studies have shown a lack of correlation between MAP and actual user performance [29].

Another popular rank-based metric is Normalized Discounted Cumulative Gain (NDCG). The advantage of NDCG is that it can consider graded relevance values, whereas MAP can only be used for binary relevance judgment. DCG at position k is defined as

$$\mathrm{DCG}\ @k = \sum_{i=1}^{k} \frac{2^{rel(q,d_i)} - 1}{\log_2(1+i)}$$

If it is binary relevance judgement, the numerator simply becomes 0 or 1. The gain at position k decays in a logarithmic fashion like below.

$$\mathrm{DCG} = rel(q, d_1) + \frac{rel(q, d_2)}{\log_2 2} + \frac{rel(q, d_3)}{\log_2 3} + \ldots \frac{rel(q, d_k)}{\log_2 k}$$

Since DCG is the sum of gains, it increases as the list contains many relevant documents. For that reason, it needs to be normalized. NDCG is the normalized version of DCG defined as follows.

$$\mathrm{NDCG}\ @k = \frac{\mathrm{DCG}\ @k}{\mathrm{IDCG}\ @k}$$

where IDCG is the maximum gain of the list, so NDCG is in the closed real interval $[0, 1]$. Figure 2.1 shows the distribution of clicked positions extracted from search logs of a real search application and DCG at position k. As we can see, their shapes are similar, indicating that NDCG is a more suitable metric for the application.

Figure 2.2 shows an example of how NDCG is calculated. In this search session, two documents are clicked. The ideal list has these positive documents at the top. So, the sum of the gain is 1.631. The actual clicked documents are in the 4th and 14th positions, and its sum of the gain is 0.681. NDCG divides DCG by the ideal DCG. Therefore, the NDCG@20 of the actual list is 0.421. The *pred* column shows the output of a model. Documents are sorted by score, positive documents move to the 2nd and 3rd position. Therefore, the NDCG@20 of the prediction is 0.693.

Other than the metrics I introduced here, there are various metrics, such as cost-based metrics and goal-sensitive metrics. Just as ranking models need to evolve follow-

query	doc_id	label	pred
evaporated milk for ice cream	152073	0	0.0003
evaporated milk for ice cream	300484	0	0.0000
evaporated milk for ice cream	46618	0	0.9980
evaporated milk for ice cream	343058	1	0.8647
evaporated milk for ice cream	180026	0	0.0003
evaporated milk for ice cream	203539	0	0.0000
evaporated milk for ice cream	261582	0	0.7906
evaporated milk for ice cream	337329	0	0.3188
evaporated milk for ice cream	312550	0	0.0000
evaporated milk for ice cream	163779	0	0.0000
evaporated milk for ice cream	120999	0	0.0269
evaporated milk for ice cream	22138	0	0.0008
evaporated milk for ice cream	266595	0	0.0013
evaporated milk for ice cream	313026	1	0.9939

	Ideal	Actual	Pred
Position of 343058	1st	4th	3rd
Position of 313026	2nd	14th	2st
Gain of 343058	1.0 (1st)	0.431 (4th)	0.5 (3rd)
Gain of 313026	0.631 (2nd)	0.256 (14th)	0.631 (2nd)
Sum of Gain	1.631	0.681	1.131
NDCG@5	1.0	0.418	0.693
NDCG@20	1.0	0.421	0.693

Figure 2.2: Example of how NDCG is calculated

ing new requirements, metrics need to be updated as well. For example, existing metrics are designed for a homogeneous list; it cannot cope with heterogeneous search results in which the system shows multi-modal results. Some researchers have attempted to develop user-focused metrics for such complex systems by introducing a user-browsing model [2].

2.4 Probabilistic retrieval

Probabilistic information retrieval models exploit reasoning provided by probability theory to estimate how likely a query is relevant to a document. This approach was promising because information retrieval systems guess whether a document satisfies the query under an uncertain understanding of the user query.

The idea of ranked retrieval was proposed by Luhn [45], and Maron, Kuhns, and Ray tested the idea that assigns a score that indicates the relevance of each document to a given query. Then top-ranked documents are returned to the user. At first, keywords were manually assigned to a set of documents and weighted based on the keywords' importance to the documents using a probabilistic approach. In the same year, Luhn suggested that word frequency and distribution in a document represents a useful indicator of word importance [46]. This approach became known as term frequency weighting

and has been continuously studied.

Sparck Jones complemented Luhn's term frequency (TF) weighting [36]. This paper introduced the idea of inverse document frequency (IDF) which states that the frequency of occurrence of a word in a document set was inversely proportional to its importance. The combination of these two weights (TF-IDF) has been widely adopted and is still used in many NLP tasks.

In 1977, Stephen Robertson defined the probability ranking principle [62], stating that the overall effectiveness of the information retrieval system will be maximized when the information retrieval system ranks and shows the documents in the collection in order of decreasing probability of relevance [10]. Since the original probabilistic model did not consider term frequency weights, many methods were proposed to effectively incorporate term frequency. The variations of TF-IDF weighting schemes were proposed in this era. This work led to the ranking function BM25 [63], which is the most commonly used scoring function nowadays.

The basic idea of BM25 is to rank documents by the log-odds of their relevance. It is based on the probability ranking principle described above. A question arises here: how can we define the probability that a document is relevant to a query? There are various answers to this question, but the most representative one commonly believed is as follows.

$$p(rel = 1|d, q)$$

where d is a document, q is a query, and rel is a boolean value indicating relevance judged by a user. Let rel is $rel = 1$ and $\bar{rel}$ is $rel = 0$ then

$$p(rel|d, q) = 1 - p(\overline{rel}|d, q)$$

From Bayes' theorem, the above formula can be represented as below.

$$p(rel|d, q) = \frac{p(d, q|rel)p(rel)}{p(d, q)}$$

and

$$p(\overline{rel}|d, q) = \frac{p(d, q|\overline{rel})p(\overline{rel})}{p(d, q)}$$

Take the log-odds ($logit(p) = log(\frac{p}{1-p})$) and apply Bayes' theorem.

$$log\frac{p(rel|d,q)}{1-p(rel|d,q)} = log\frac{p(d,q|rel)p(rel)}{p(d,q|\overline{rel})p(\overline{rel})} \tag{2.1}$$

$$= log\frac{p(d|q,rel)}{p(d|q,\overline{rel})} + log\frac{p(rel|q)}{p(\overline{rel}|q)} \tag{2.2}$$

Since the term $log\frac{p(rel|q)}{p(\overline{rel}|q)}$ does not affect scoring because it is independent of d, we get

$$log\frac{p(d|q,rel)}{p(d|q,\overline{rel})}$$

This formula is the core of the probabilistic retrieval model. For simplicity, the binary independence model assumes that

- Given relevance, terms are statistically independent.
- The presence of a term in a document depends on relevance only when that term is present in the query.

because modeling documents as a set of words is more convenient. Then,

$$\sum_{t\in(q\cap d)} log\frac{p(d_t=1|rel)p(d_t=0|\overline{rel})}{p(d_t=1|\overline{rel})p(d_t=0|rel)}$$

where $t \in (q \cap d)$ represents terms appearing in both the query and document.

BM25 is an extension of this binary independence model. Though BM25 is a family of ranking models, the most common implementation of the model is that given a query q, containing terms $t_1, ..., t_M$, the BM25 score of a document d is computed as

$$f_{BM25}(d,q) = \sum_{t_q\in q} \frac{\mathrm{IDF}(t_i)\cdot \mathrm{TF}(t_i,d)\cdot(k_1+1)}{\mathrm{TF}(t_i,d)+k_1\cdot(1-b+b\frac{\mathrm{LEN}(d)}{\mathrm{avgdl}})}$$

where $\mathrm{TF}(t,d)$ is the term frequency of t in document d, $\mathrm{LEN}(d)$ is the length (number of words) of document d, and avdl is the average document length in the text collection from which documents are retrieved. k_1 and b are free parameters, $\mathrm{IDF}(t)$ is the IDF weight of the term t. The idea of TF-IDF is that terms appearing frequently are consid-

ered significant while terms distributing across documents are considered common and less important.

2.5 Vector space model

As an alternative approach for ranked retrieval, Gerard Salton introduced the Vector Space Model in 1975 [64]. The idea is to represent both queries and documents as vectors in a latent Euclidean space and rank documents according to the query's similarity. The most common method for measuring the similarity between two vectors is to compute the cosine similarity of the two vectors denoted as follows.

$$f(q, d) = \frac{\mathbf{v}_q \cdot \mathbf{v}_d}{|\mathbf{v}_q|\,|\mathbf{v}_d|}$$

where the numerator represents the dot product of the query and document vectors, while the denominator is the product of their Euclidean lengths. The dot product $\mathbf{x} \cdot \mathbf{y}$ of two vectors is defined as $\sum_i x_i y_i$. Euclidean length is defined as $\sqrt{\sum_i \mathbf{v}_i}$.

The most well-known method of the vector space models is Latent Semantic Indexing (LSI), which uses Singular Value Decomposition (SVD) to reduce the dimensionality of the vector space of a document set [20]. More research on learning representations of queries and documents has been done after the breakthrough of neural networks. This method later became called the representation-based approach.

In practice, computing the cosine similarity between a given query vector and each candidate document in the search space, sorting the documents by score, and selecting the top k documents can be costly to process user requests in real time. For that reason, the Approximate Nearest Neighbor (ANN) search is usually employed, and there are many libraries to do fast retrieval for vectors.

2.6 Limitation of traditional methods

Up to this point, the scoring functions were manually devised and tuned by hand through experimentation. However, it is almost impossible to satisfy all diverse information needs, and hence, what users want is not reflected in the relevance or similarity employed in the above methods. In the next chapter, I discuss learning to rank, which

updates parameters of functions by using relevance feedback from users.

Chapter 3

Conventional Learning to Rank

BM25 has only two parameters to adjust. Researchers were usually able to maximize the utility of the system through grid search, but recently documents have become more complex and dynamic in which the vast number of parameters are needed to model relevance.

Learning to rank is a task to automatically adjust the parameters of ranking models using training data, such that the model can rank new documents with predicted relevance scores. Learning to rank uses relevance labels, typically click data as implicit feedback, to train models. This is the difference from the models in the previous chapter.

This new field began to be explored in the late 1980s [72, 24, 26], but those attempts were not very successful until the 2000s. This is because traditional ranking functions in information retrieval used a tiny number of features such as term frequency, inverse document frequency, and document length. It was possible to tune weights by hand. Besides, the amount of data used for training was limited, meaning that it was challenging to collect search logs from real user needs.

In the 2000s, as more people had access to the Internet, the amount of potential training data increased, making it possible to leverage machine learning techniques to build effective ranking models. A new area of research called learning to rank gained popularity during this period.

Learning to rank is a supervised learning task that has training and testing phrases. Most learning to rank models learn the ranking functions by minimizing loss functions based on signals representing the relevance between queries and documents. The pro-

cess of learning to rank is as follows. In training, many documents to rank are given, each document consisting of features and labels. Features used in learning to rank can often be categorized as *query-dependent features* (e.g., BM25), *query-level features* (e.g., query length), and *query-independent features* (e.g., incoming link count and document length). Then the parameters of the ranking model are updated to minimize a certain loss function on the training data. In testing, given a new set of features, the ranking model is applied to produce a ranked list which is supposed to maximize the utility. Depending on the loss function's design, learning to rank models can be grouped into three types: pointwise, pairwise, or listwise approach.

3.1 Pointwise approach

The pointwise approach looks at a single query-document pair $\langle q, d_i \rangle$ at a time in the loss function and learns the ranking function that outputs the relevance degree of the document. Such a function is usually called a scoring function. Based on the scoring function, the system ranks documents and produce the final ranked list. The loss function examines the accurate prediction of the ground truth label for every single query-document pair. In this approach, the ranking task is simplified and transformed into classification, regression, or ordinal classification. The examples of the pointwise approach are Subset Ranking [14], McRank [41], Prank [16], and SVM for Ordinal Classification [67]. McRank employs the cross entropy-loss with softmax over categorical labels Y defined as follows.

$$\mathcal{L} = -log\left(p\left(y_{q,d} \mid q, d\right)\right) = -log\left(\frac{e^{s_{y_{q,d}}}}{\sum_{y \in Y} e^{s_y}}\right)$$

where, $s_{y_{q,d}}$ is the score of the model for label $y_{q,d}$.

This approach was superior to non-machine learning models in the sense that parameters were updated throughout the learning process, but it was thought to be less effective in ranking tasks because pointwise loss functions do not take into account the inter-dependency between documents, meaning that its loss function cannot utilize the information of the relative position of the document in the final ranked list. For that reason, better ranking paradigms that directly maximize the utility of the final ranked list based on pairwise loss functions and listwise loss functions have been proposed.

3.2 Pairwise approach

In the pairwise approach, given $\langle q, d_i, d_j \rangle$, the model gives a higher score for a more relevant document. Therefore, the pairwise approach does not focus on accurately predicting the relevance degree of each document but care more about their relative position, which is distinct from the previous approach. In this sense, it is closer to the concept of "ranking" as the goal of training is to minimize the number of inversions in ranked documents while the pointwise approach just outputs the score of the single document. The pairwise ranking task is usually reduced to a classification task on document pairs. The examples of the pairwise approach include Ranking SVM [35], RankBoost [23], RankNet [7], LambdaRank [8], and LambdaMART [9]. Pairwise losses generally have the following form.

$$\mathcal{L} = \phi\left(s_{q,d_i} - s_{q,d_j}\right)$$

I take popular pairwise losses as examples and describe them in detail.

3.2.1 Pairwise hinge loss

The objective is to learn representations with a small distance between them for positive pairs, and greater distance than some margin value for negative pairs. The pairwise hinge loss is defined as follows.

$$\mathcal{L} = \max\left(0, \varepsilon - \mathrm{rel}\left(q, d^+\right) + \mathrm{rel}\left(q, d^-\right)\right)$$

where ε is the parameter determining the margin of hinge loss. The hinge loss is also known as Max-Margin Loss.

3.2.2 Pairwise cross-entropy loss

The loss function is based on the entropy theory defined in the differences in scores of pairs of documents as follows.

$$\mathcal{L} = -log\left(p\left(d^+ \mid q\right)\right) = -log\left(\frac{e^{s(q,d^+)}}{\sum_{y \in Y} e^{s(q,d)}}\right)$$

Logistic Loss and Multinomial Logistic Loss are other names for the cross-entropy loss. Note that learning to rank models typically considers few negatives candidates since computing the softmax over the full collection is prohibitively expensive.

The pairwise approach is more suitable than the pointwise approach for the ranking objective. However, the pairwise approach takes into account only the pair document dependence, which means that dependence between each document in the whole candidate set is still not fully considered.

3.3 Listwise approach

The listwise approach constructs the loss functions that directly maximize the effectiveness of the ranked list. The difference from the pairwise approach is that the listwise models consider the ranking loss with each query and their candidate document list together. Therefore, the property of ranked list that documents are ordered by relevance score is maintained and ranking evaluation measure can be more directly incorporated into the loss functions in learning.

The listwise learning to rank models can be further divided into two types: direct optimization of measures of information retrieval such as SoftRank [69] and AdaRank [73], and minimizing a loss function that is defined based on understanding the unique properties of the ranking such as ListNet [11] and ListMLE [39].

ListNet computes the probability distribution over all possible permutations based on model score and ground truth labels. The loss is then given by the KL divergence between these two distributions. ListMLE utilizes the likelihood loss of the probability distribution based on Plackett-Luce model for optimization.

Widely used measures in information retrieval such as NDCG and MAP obviously differ from the loss functions described above sections. In such a situation, a natural question to ask is whether the minimization of the loss functions can lead to the optimization of the ranking measures. Researchers have tried to proved the connection between the metrics and the loss, and it has shown that the regression and classification based losses used in the pointwise approach are upper bounds of (1-NDCG) [41, 12, 5].

Though it is known that listwise approaches are more desirable for learning to rank because of the nature of the design, as shown in [6], pointwise and pairwise approaches tend to be preferred for performance reasons in practice [79].

Chapter 4

Neural Information Retrieval

Neural networks recently have demonstrated impressive performance in various machine learning tasks, including information retrieval. Neural ranking models [1, 52] have continuously broken state-of-the-art records. This is thought to be because neural models can learn more complex interactions between queries and documents, whereas conventional learning to rank models rely on hand-created features. Neural ranking models embed queries and documents into a low-dimensional latent space and computes the relevance score against a query-document pair. Since the emergence of the first successful neural ranking model in 2013 [32], researchers have proposed many neural models to information retrieval tasks.

4.1 Text Representation

In document ranking, it is not an exaggeration to say that how to represent text determines the characteristics of the ranking, and the ability of learning text representation makes neural ranking models distinct from conventional learning to rank models. In this section, I introduce several assumptions and approaches for text representation.

Documents and queries are a sequence of words. For computers to process text, it must be converted into some convenient format while preserving the linguistic information. In the early days, text was represented in a one-hot vector, where one bit corresponded to a unique term with no feature/bit sharing. This method did not take into account word order, so the relationship between words was disregarded. For ex-

ample, ”hot dog” and ”dog hot” have exactly the same representation. Because of this nature, this approach was classified as bag-of-words.

As aforementioned, the binary independence model assumes that terms are statistically independent. This means that the same documents are retrieved in exactly the same order for the queries ”hot dog” and ”dog hot”. Conventional models are based on this assumption, and BM25's effectiveness has shown in the history of information retrieval, but how plausible is the assumption? Several papers have questioned this term independence assumption.

Another assumption made by the binary independence assumption is that the presence of a term in a document depends on relevance only when that term is present in the query” does not admit the existence of misspellings and synonyms, causing a lexical gap between information needs and actual retrieved documents. For example, users expect that queries such as ”UK prime minister” and ”high-end phone” match terms that are lexically far but semantically similar. It has also been proved that methods relying on term frequency are not sufficient for short text retrieval because the term frequency distribution becomes flat, making capturing word significance infeasible.

The vector space model has attempted to address this issue and is therefore called distributed representation in contrast to one-hot representation (local representation). The breakthrough happened in 2013. Milkolov et al. proposed a new way, so-called Word2Vec, to embed words in a low-dimensional space using a neural network [49].

Since text consists of words, how to represent sentences using word embeddings has been extensively discussed. The simplest way is to take the sum or average of all word vectors of the document. This method is fast, but it still disregards word order. The Long Short-Term Memory (LSTM) was the most commonly used network architecture to capture inter-term dependencies. Convolutional Neural Network (CNN) grew in parallel to LSTMs and showed its effectiveness but did not attain the popularity of LSTMs since they cannot capture word order on their own. The introduction of Transformers [70] was another revolution. The transformer is an architecture that can capture long-range word dependencies with no recurrence but employs the attention mechanism. Despite this simple architecture, transformer-based models have significantly outperformed existing models.

These techniques were invented for challenges in Natural Language Understanding. The question here is whether it also improves the performance of information retrieval

systems. Most recently, Mitra et al. have shown that no significant loss is observed in the models that incorporating the query term independence assumption in web search [51]. Another paper has reported that state-of-the-art BERT-based models are mediocre in product search [66]. These may suggest that capturing inter-term dependencies is not critical in some search applications, whereas it is a central concern in question answering.

4.2 Types of ranking models

According to the existing literature [32], the core problem of document ranking can be formulated essentially as text-matching as follows: Given two texts T_1 and T_2, a scoring function that takes the representation of each text computes the degree of relevance like below.

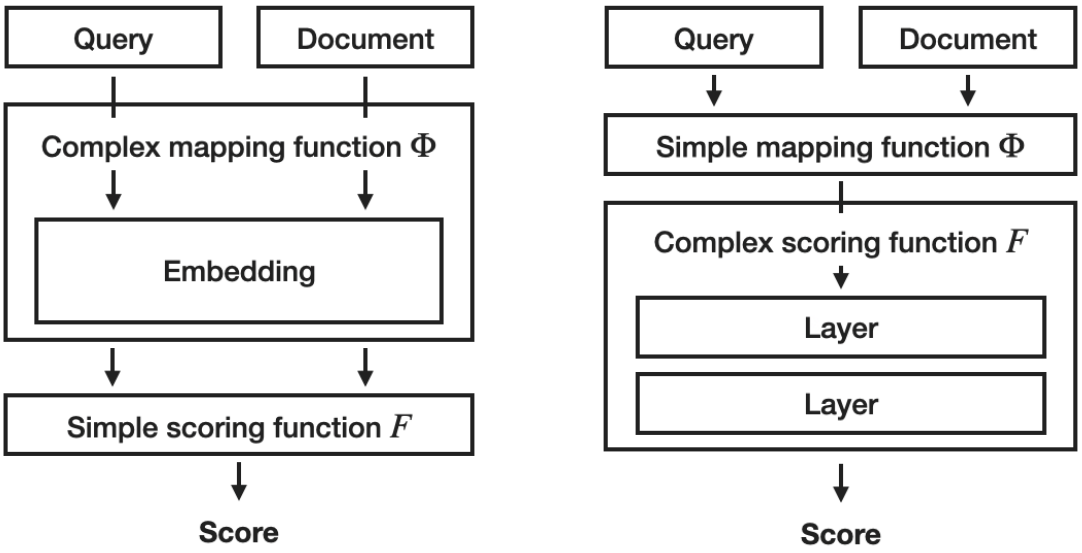
$$match(T_1, T_2) = F(\Phi(T_1), \Phi(T_2)),$$

where Φ is a mapping function to convert each text into a representation vector, and F is the scoring function that captures the interactions between them. Depending on the choice of the two functions, existing neural ranking models can be categorized into two types of architectures, namely representation-based model and interaction-based model.

4.2.1 Representation-based models

Models in this type embed both queries and documents into a latent low-dimensional space separately and assess their similarity based on such dense representations. In this approach, Φ is a complex representation mapping function, while F is a relatively simple matching function such as cosine similarity. Because of its characteristics, models in this type are also referred to as a two-tower model or a dual-encoder model.

The underlying assumption of this type of model is that 1. queries and documents have some similarities, and 2. engagement data represents the similarity relations between them. Latent space models bridge the semantic gap between words through reducing the dimensionality of latent space from term level matching to semantic matching and correlating semantically similar terms. The mapping function is automatically



(a) Representation-based model (b) Interaction-based model

Figure 4.1: Representation-based and interaction-based architectures

learned from data. Examples in this regard include DSSM [32], ARC-I [31], and DESM [50].

DSSM maps text to a common semantic space with a neural network. The first layer applies a letter n-gram based word hashing to reduce the dimensionality of feature vectors. The final layer computes the cosine similarity between the calculated vectors as a measure of their relevance.

ARC-I stacks 1D convolutional layers and max-pooling layers are applied to the input texts to produce their high-level representations. ARC-I then concatenates the two representations and uses an MLP as the evaluation function.

Representation-based models usually employ a Siamese architecture that consists of a stack of fully-connected layers to learn semantic representations of queries and documents, which are then used to compute the relevance score between them with some similarity measure. It mitigates the issue of a lexical gap.

However, this approach has its limitation because of the nature that each representation is formed without knowledge of the others, but instead, both query and document vectors can be precomputed, which allows us to apply approximate nearest neighbors at query time. This property is ideal for production environments whose performance

requirements are rigorous.

4.2.2 Interaction-based models

Interaction-based models learn explicit interactions between pairs of queries and documents. This allows direct modeling of exact or near-exact matching terms (e.g., synonyms), which is crucial for relevance ranking. In this approach, Φ is usually a simple mapping function, while F becomes more complex. Since relevance matching is fundamentally a matching task, most recent neural architectures, such as ARC-II [31], DRMM [27], PACRR [33], adopt an interaction-based design. They operate directly on the similarity matrix obtained from products of query and document embeddings and build sophisticated modules on top to capture additional n-gram matching and term importance signals.

ARC-II learns directly from interactions rather than from individual representations. A first convolutional layer creates combinations of the inputs via sliding windows on both sentences so that the remaining layers can extract matching features. ARC-II defines an interaction function by computing the similarity between every n-gram pair. After that, several convolutional and max-pooling layers are leveraged to obtain the final relevance score. It was shown in [31] that ARC-II has a superior retrieval accuracy than ARC-I, a representation-based architecture.

DRMM builds interactions between pairs of words from a query and a document, and subsequently creates a fixed-length matching histogram for each query term. The final score is calculated by a weighted aggregation of the score of the query terms.

4.2.3 Hybrid models

Relevance matching models such as DRMM and KNRM resemble traditional information retrieval measures in that they directly consider the similarity of the document for the query. On the other hand, many NLP problems, such as question answering and textual similarity measurement, require more semantic understanding and contextual reasoning than specific term matches. In this context, applying contextualized language models to information retrieval has been studied. [55] employed pre-trained language models such as ELMo [57] and BERT [21], that have achieved impressive results on

various natural language benchmarks, showed that this approach potentially works in information retrieval tasks as well.

Since such search applications have to consider those different signals, hybrid methods have been proposed, such as DUET [27] and HCAN [59]. DUET combines signals from a local model for lexical matching and a distributed model for semantic matching. HCAN has a similar architecture that minimizes the losses of both lexical and semantic modules jointly.

4.3 Matching with multiple document fields

Existing ranking models typically assume that documents to retrieve consist of one text field (e.g., passage re-ranking). A few studies have discussed how to use evidence from structure to improve the performance of information retrieval systems. Wilkinson proposed several heuristic methods of combining section-level and document-level evidence, such as taking the maximum section score or taking a weighted sum of section scores [71].

In non-neural models, Robertson et al. proposed BM25F [61], an extension to the original BM25 model. They discussed that simply taking into account the linear combination of field-level scores is not enough for multiple-field document ranking tasks because it bypasses the careful balance across query terms in the BM25 model. The BM25F solution is first to combine frequency information across fields on a per-term basis, then compute a retrieval score using the balanced BM25 approach.

There are some alternative approaches to BM25F for the multiple-field document retrieval task. Piwowarski and Gallinari proposed a model based on Bayesian networks for retrieving semi-structured documents [58]. Svore and Burges proposed a supervised approach, called LambdaBM25 [68], based on the attributes of BM25 and the training method of LambdaRank. LambdaBM25 can consider multiple document fields without resorting to a linear combination of per-field scores. Kim and Croft introduced a framework for estimating field relevance based on the combination of several sources [38].

Regarding multiple-field document retrieval in neural models, Zamani et al. studied some of the principles for designing neural architectures for ranking documents with

multiple fields [78]. Their model, called NRM-F, formulates the document representation learning function Φ_D as follows.

$$\Phi_D\left(F_d\right) = \Lambda_D\left(\Phi_{F_1}\left(F_1\right), \Phi_{F_2}\left(F_2\right), \cdots, \Phi_{F_k}\left(F_k\right)\right)$$

where Φ_{F_i} denotes the mapping function for the field F_i and Λ_D aggregates representations learned for all the fields. Λ_D simply concatenates the input vectors to be served in the matching function. Then, a stack of fully-connected layers outputs the final retrieval score.

In NRM-F, both query text and text fields are represented using a character n-gram hashing vector, introduced by Huang et al. [32]. Then, a convolution layer is employed to capture the dependency between terms.

NRM-F considers the interactions between query and fields. For example, if the document has three fields: url, title, and body, the feature pairs are $query \odot url$, $query \odot title$, and $query \odot body$ where $\odot$ denotes the Hadamard product of the representation; which is the element wise product of two matrices with the same dimensionality.

They have found that learning separate latent spaces for matching a query against different document fields is more effective than having a embedding space shared across all fields. This has been thought that the different document fields may contain information relevant to different aspects of the query intent. They also state that the ranker should score the whole document jointly, rather than generate a per-field score and aggregate, which is different from popular methods in recommender systems described in the next subsection.

NRM-F was designed for web search as the paper was written by researchers in Microsoft. There may be a better architecture for the *other industrial search applications* I am targeting. I discuss the differences in architecture in section 4.6.

4.4 Information retrieval and recommendation

Information retrieval and recommendation have become two separate fields, but they share many things in common. For instance, both tasks' goal is to generate the best permutation that maximizes the utility of an ordered list of documents (or items) for a given query (or user features, also called context). In addition, standard information retrieval

evaluation metrics and losses are frequently used by the recommendation community too.

Single-field document ranking models focus on capturing text matching patterns, and recommendation models concentrate on capturing interactions between context and items, which usually consist of multiple fields. I hypothesized that multi-field document ranking is located somewhere between single-field document ranking and recommendation because multi-field document ranking has both properties. If this hypothesis is correct, we could use techniques for capturing text significance in the information retrieval community and methods for combining multiple evidence in the recommendation community, which is beneficial for both communities.

This is not the first time to discuss the similarity between information retrieval and recommendation. Belkin and Croft discussed the same topic decades ago [3]. They concluded that their underlying goals are essentially the same. Costa and Roda reformlated the recommender systems problem as an information retrieval one [15].

Zamani et al. has shown that jointly learning both an information retrieval model and a recommendation model substantially outperforms models the retrieval and recommendation models trained independently [76]. It indicates that approaches employed in the literature of recommender systems could potentially work in information retrieval tasks.

4.4.1 Factorization machines

While only a few studies on dealing with multiple fields have been done in information retrieval, this is a much more popular topic in the recommendation community as items to recommend naturally consist of multiple fields.

Recent recommendation models in the community are based on the factorization machine proposed by Rendle in 2010 [60] that was supposed to overcome the issues Support Vector Machines (SVMs) have. A distinct advantage of FMs is that it allows parameter estimation under very sparse data where SVMs fail. The factorization machine models the interactions between two fields i and j as the dot product of their corresponding embedding vectors $\mathbf{v}_i, \mathbf{v}_j$ as follows.

$$\Phi_{FMs} = w_0 + \sum_{i}^{n} w_i x_i + \sum_{i,j}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i x_j$$

where the model parameters that have to be estimated are:

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{V} \in \mathbb{R}^{n \times k}$$

and $\langle , \rangle$ is the dot product of two vectors of size k:

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle := \sum_{f=1}^{k} \mathbf{v}_{i,f} \cdot \mathbf{v}_{j,f}$$

A row $\mathbf{v}_i$ within $\mathbf{V}$ describes the i-th variable with k factors.

FMs can be viewed as interaction-based ranking models in information retrieval since FMs map features into a latent space and combine them to compute the score somehow. This basic idea has worked well in recommendation tasks and has been continuously improved until now.

The original FMs treat all interactions evenly but enumerate all cross features may lead to degraded learning performance, since some interactions may be irrelevant or redundant, introduce noise, and increase the difficulty of learning. This issue has been attempted to solve in different ways.

In 2016, the field-aware factorization machine (FFM) was proposed [37] to learn such difference explicitly by $n - 1$ embedding vectors for each feature.

$$\Phi_{FFMs} = w_0 + \sum_{i}^{n} x_i w_i + \sum_{i,j}^{n} x_i x_j \langle \mathbf{v}_{i,F(j)}, \mathbf{v}_{j,F(i)} \rangle$$

where $F(i)$ is a field x_i belongs to and $\mathbf{v}_{i,F(j)}$ is a vector $F(j)$ corresponding to field x_i. The paper has demonstrated that FFMs outperform three well-known models, LM, Poly2, and FM. However, FFMs can overfit easily because of its large number of parameters. To address the issue, the field-aware probabilistic embedding neural network (FPENN) [44] was proposed in 2018. Various confidence levels for feature embeddings are incorporated to enhance robustness and accuracy. Applying the variational autoencoder to recommendation tasks was also proved effective in grocery recommendation [48].

Since an alternative approach to prevent the overfitting issue, the field-weighted factorization machine (FwFM) [56] was proposed in the same year, arguing how to reduce the number of parameters of field-aware factorization machines. The idea of the field-aware factorization machine is based on the assumption that how interaction

affects varies by field pair. The field-weighted factorization machine redefined this issue to simply estimate the explicit weight of field pairs.

$$\Phi_{FwFMs} = w_0 + \sum_{i}^{n} x_i w_i + \sum_{i,j}^{n} x_i x_j \langle \mathbf{v}_i, \mathbf{v}_j \rangle r_{F(i),F(j)}$$

where $r_{F(i),F(j)}$ is a weight of the interaction strength between $F(i)$ and $F(j)$.

Since the attention mechanism is known to address such issues, the interaction-aware factorization machine [30] has introduced an attention network to learn feature interaction importance, which simultaneously considers field information as auxiliary information, instead of learning explicit weights.

4.5 Examples of industrial application

Nowadays, information retrieval and recommendation are studied actively by researchers in the industry because it directly affects the user experience. In this subsection, I introduce some papers published from the industry.

Airbnb shared the success stories of applying machine learning to their search product [28]. The very first implementation of their search ranking was a manually crafted scoring function. They started with a gradient boosting decision tree model to replace the manual scoring function. It improved the performance, but the gains in online bookings saturated. This made the moment ripe for trying sweeping changes to the system. They eventually trained a neural network using outputs from gradient boosting decision trees and a factorization machine along with other features. They also shared unsuccessful attempts. For example, they reported that embedding listing IDs did not work despite its success in other applications.

Industrial search applications have a common issue of filling the gap between users' information needs and the actual query. In Amazon, they conducted a research project focusing on semantic matching using neural networks [54]. Pure lexical matching via an inverted index falls short in the respect due to several factors: (1) lack of understanding of hypernyms, synonyms, and antonyms, (2) fragility to morphological variants (e.g. "woman" vs. "women"), and (3) sensitivity to spelling errors. According to some estimates of web search logs [19], 10-15% queries are misspelled, causing confusion and frustration. Their model employed average pooling in conjunction with n-grams to fill

the semantic gap. The letter n-gram approach is said to reduce vocabulary, generalized to unseen words, and robust to misspelling and inflection.

In YouTube, they described their large scale multi-objective ranking system for recommending what video to watch next [79]. Typical recommendation systems follow a two-stage design with a candidate generation and a ranking because re-rankers are usually computationally expensive. Scalability is crucial because the YouTube recommendation system is being built for billions of users and videos. The model must be effective at training and efficient at serving. For that reason, they chose a pointwise approach that is simple and efficient to scale to a large number of candidates, while pairwise and listwise approaches need to score pairs or lists multiple times to find the optimal ranked list give a set of candidates.

4.6 Categorizing search applications

This paper discusses what architecture is effective for modern search applications, but the optimal model may vary from application to application due to each system's different characteristics. However, categorizing applications may help develop hypotheses when designing a ranking model. I list the attributes of search applications from various perspectives.

4.6.1 Keyword matching vs. free text matching

This is the perspective of user input. In keyword matching, users enter a set of words as a query like "fridge black affordable". In contrast, free text matching systems need to accommodate question-like queries such as "Why won't my cat stop bothering me while I'm trying to work?". In the former case, the query is sampled from a document, and the words in the query can be considered as having no interdependencies. It has been known that bag-of-words methods like average pooling works in such systems (e.g. [54, 42]). In the latter case, queries and documents are heterogeneous. Imagine a question answering-like system, users expect to get answers to their questions, but they do not expect to see the questions to their questions. In a sense, we can hypothesize that models that assess the similarity between queries and documents may not be appropriate, but contextualized language models that can deal with synonymy and ambiguity may show

	Vocabulary size
Recipe search	229,559
Web search	3,213,845
Character trigram	26 letters ^ 3 grams = 17,576

Table 4.1: Comparison of vocabulary size

better performance. Google announced that they started using a contextualized language model, BERT [21], in their search engine to understand the context of the query better [1].

4.6.2 Long text vs. short text

This is the perspective of the document length that directly affects the richness of available information. Information retrieval systems deal with text at varying levels of length from single phrase to documents containing thousands of words. The length of the document affects the ease of capturing textual information. Methods that rely on term frequency, such as BM25 may not capture significance well for shorter texts because the relevance score is calculated based on the number of words contained in the target text field. For example, given the query ”paella” and getting 1,000 results, it is hard to distinguish which recipe is more relevant because there are no recipes that repeat ”paella” in the title.

Relating to this topic, Cohen et al. examined how granularity affects the performance of neural ranking models [13]. Their experiment has shown that when candidate answers are short, CNNs and LSTM perform at equivalent levels with differences attributed to attention methods and structure differences beyond the convolutional and LSTM layers.

4.6.3 General vs. domain specific

This is the perspective of the document content. Hotel search, job search, recipe search, ... these systems focus on searching a specific domain of information. Their vocabulary size is limited, unlike information retrieval systems that deal with general topics such as web search and question answering. The larger the vocabulary size, the more

[1] https://www.blog.google/products/search/search-language-understanding-bert/

Perspective	Web search	*Industrial search application*
User input	Keyword & Free text	Keyword
Text length	Long	Short
Vocabulary	General	Domain specific
Search space	Re-ranking	Re-ranking

Table 4.2: Differences of properties by application

challenging it is to put in memory, affecting the design decision. Table 4.1 compares vocabulary sizes of different services (Cookpad as a recipe search and MS MARCO as a web search) and an encoding method. Character trigram fits into a fixed size, no matter how large the vocabulary is, but it tends to perform poorly compared to word-level representations.

4.6.4 Ranking vs. re-ranking

This is the perspective of the search space. In the re-ranking task, models determine which documents are most likely to be engaged from the set of candidates that have already been filtered by text features, whereas documents are more textually random in the ranking task. In this case, re-ranking models put more importance on other features besides lexical matching signals, and being trained with hard examples would result in better performance.

4.6.5 Web search and other industrial search applications

Web search has been studied much since the early day of information retrieval. Several neural ranking models for web search already exist but the form of the application I am discussing in this thesis is slightly different from web search as shown in Table 4.2. A web page contains thousands of words, and the topics that web search engines cover are incredibly diverse. In contrast, industrial search applications usually deal with a specific topic, and the document is much shorter in length than web pages. We need to absorb these changes in architecture.

4.7 Validity of experiments on neural models

Neural networks have become the method of choice for people working on information retrieval and recommender systems. With the growing interest in machine learning, it has become challenging to keep track of what represents the *state-of-the-art* methods as new techniques become stale too fast. Some researchers express skepticism that neural network models are representing genuine advances in effectiveness. Recently, researchers have conducted meta-analyses in different fields and reported shocking results on the hype surrounding neural approaches.

In the field of information retrieval, Lin first discussed the topic [43]. He provided anecdotal evidence that two recent neural information retrieval models demonstrate "wins" by comparing against weak baselines on TREC 2004 Robust Track (Robust04 for short). In the same year, more rigorous meta-analysis has been reported by Lin et at. [75]. They examined 130 papers from 2005 to 2018 in the following venues to identify those that reported results on Robust04: SIGIR, CIKM, WWW, ICTIR, ECIR, KDD, WSDM, TOIS, IRJ, IPM, and JASIST. 4 out of the 5 models examined were not able to significantly beat the baseline, suggesting that gains attributable to neural approaches are not as widespread as the literature suggests. The analysis was conducted using the Answerini toolkit [74] and MatchZoo [22]. They compared non-neural methods: BM25, Query likelihood with Dirichlet smoothing (QL), and RM3 with five neural models: DSSM, CDSSM, DRMM, KNRM, and DUET were selected. Effectiveness was measured on five-fold cross-validation and statistical significance of metric differences was assessed using a paired two-tailed *t*-test. They reported that only DRMM showed a statistical improvement ($p = 0.0032$) and concluded that at least some of the gains reported in the literature are illusory, while neural networks no doubt represent an exciting direction in information retrieval.

A meta-analysis has been done in recommender systems with the same motivation [17]. They attempted to assess the effectiveness of the models from recent papers. They considered 18 algorithms that were presented at top-level research conferences. Only 7 out of them could be reproduced with reasonable effort. However, for these methods, it was tuned out that 6 of them can often be outperformed with comparably simple heuristic methods such as nearest-neighbor. The remaining one clearly outperformed the baselines but did not consistently outperform a well-tuned non-neural linear ranking

method.

Besides difficulties in reproducibility and weak baselines, Fuhr further discussed the validity of performance measurement [25]. He is concerned that some widely used metrics such as Mean Reciprocal Rank (MRR) are not interval scales, so it is impossible to compute a mean or apply a significance test that requires an interval measure. As an improvement idea, he suggests regarding effect size d denoted as follows.

$$d = \frac{\mu_1 - \mu_2}{\sigma}$$

where μ is the mean and σ is the variance. He also pointed out that some papers published at SIGIR conducted multiple testing without correction. He reported 5 out of 8 papers nominated for the best paper award at SIGIR 2017, 12 out of 12 learning to rank papers at SIGIR 2018, and 3 out of 9 papers nominated for the best paper award at SIGIR 2020 had this problem.

Chapter 5

Neural Ranking Models for Multiple Field Documents

Single-field document ranking has long been a central topic in information retrieval, and many ranking models have been proposed in the history. However, little is known about how to design ranking models for multi-field documents. I hypothesized that multi-field document ranking is located somewhere between single-field document ranking and recommendation. In fact, Zamani et al. has shown that joint modeling of search and recommendation substantially outperforms the retrieval and recommendation models trained independently [76]. It motivated me to investigate whether the methods developed for recommendation tasks improve the performance of document ranking and, if not, how they differ. In the series of my experiments, I examine the following research questions.

- RQ1. Is it important to learn feature interactions?
- RQ2. Is it necessary to limit feature interactions?
- RQ3. Does feeding first-order features contribute to effectiveness?
- RQ4. Are there any other important feature interactions besides query-fields?

As aforementioned, the validity of experiments in information retrieval has recently been a concern in the community. To address the issue, I conducted experiments with

#queries	2,733,549
#unique queries	94,993
#unique presented recipes	301,078

Table 5.1: Basic Statistics of the dataset

proper statistical testing and published the implementation on GitHub [1] for reproducibility.

5.1 Dataset

There are some publicly available datasets that have been used in the community. TREC collections provide a variety of documents such as Robust, ClueWeb, Gov2, and Microblog. Recently, there has been a growing demand for large datasets to train and validate neural models that require a lot of data. Microsoft has released a large web search dataset called MS MARCO [2]. The documents consist of multiple fields. NRM-F used this dataset for training. However, web search is slightly different from the applications I am targeting, such as product search, hotel search, and recipe search. For that reason, I obtained a dataset from Cookpad.

5.1.1 Recipe search dataset

Cookpad is an online recipe community platform. Users can publish and search for recipes on the platform. The size of the dataset is shown in Table 5.1. The dataset consists of two tables: master recipe data and search logs.

Recipe data

Recipes are structured data as shown in Table 5.2. The description is a free text field; some recipes have a surprisingly long description while there are recipes with no description. The ingredients are an unordered set of entities. Figure 5.1 shows the distribution of the number of words in each field. As can be seen from the figure, text fields

[1] `https://github.com/rejasupotaro/master-thesis`
[2] MS MARCO `https://microsoft.github.io/msmarco/`

Feature	Type	Example
recipe_id	Integer	1
title	String	Honey garlic chicken thighs
description	String	This recipe has always been my favorite
ingredients	String set	chicken, salt, crushed red chilli, ...
country	String	GB

Table 5.2: The schema of recipe data

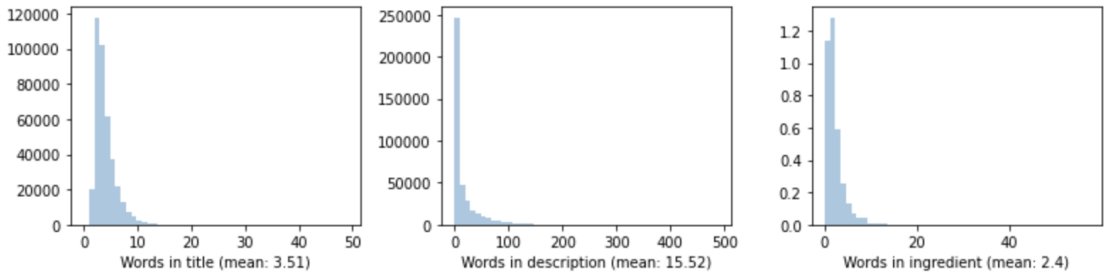


Figure 5.1: The number of words in text fields

are generally short in length, making it difficult to capture text significance using term frequency-based methods.

Search logs

A search log is an event log issued when a user clicks a recipe in the search results. The attributes of each event are listed in Table 5.3. *fetched_recipe_id* indicates what recipes were retrieved against the query and *position* shows the clicked recipe position in the list.

Feature	Type	Example
session_id	Integer	1
query	String	hot dessert
page	Integer	1
recipe_id	Integer	1
position	Integer	1
fetched_recipe_ids	String	1,2,3,4,5,6
total_hits	Integer	256

Table 5.3: The schema of search log

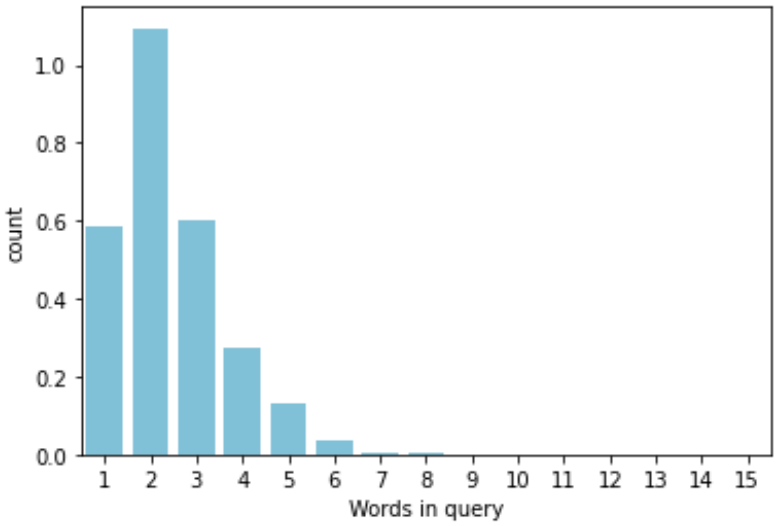


Figure 5.2: The number of words in query

Figure 5.2 shows the distribution of words in a query. Most queries contain no more than three words. Examples of long queries are ”chicken wings potato japanese style” and ”moist and fluffy sponge cake” but a small number of queries look like a question like ”how to cut water melon”.

I marked clicked recipes as positive examples. I assume that users scan recipes presented in a search results page one by one from the top of the list. The scan will continue after observing non-relevant recipes but stops after a relevant recipe is found. Motivated by this assumption, I only consider the recipes above the last clicked recipe on the list.

5.2 Data processing and modeling

Search logs are aggregated by session ID and query. Each list is cut off by the clicked position based on the assumption that retrieved recipes after the clicked position were not likely to be examined. Labels are then put by referring to the position. Now the data is composed in a listwise manner containing n positives and m negatives. All positives in the list are used. The maximum number of negatives per positive can be specified, which I set 10 in this experiment.

I chose five features: query, title, description, ingredients, and country for training.

The text embedding is shared across text fields. This data is concatenated with search logs.

To determine the direction of the modeling, I categorized this task as *keyword matching domain-specific structured document re-ranking task* based on what I discussed in *Categorizing search applications*. Regarding the text representation, I take the average of vectors to obtain fix-sized vectors as I assume that terms are almost independent. In this type of application, it is not uncommon to assume term independence. Amazon's experiment showed that taking the average of term vectors performed similar or slightly better than recurrent units with significantly less training time [54]. The countries are treated as a category and embedded into a latent space.

In my experiments, I focus on two models: NRM-F and FwFM to examine how the choice of architecture affects effectiveness. The table summarizes the differences between those two models.

	NRM-F	FwFM
First-order features	Not used	Used
Interaction selection	Query-field	All
Interaction representation	Hadamard product	Dot product
Interaction aggregation	Concatenation	Summation

5.3 Training and validation

In order to follow the standard experimental settings in the community, I examined which metrics and losses are employed in recent 8 papers (Table 5.4).

I adopt the most common metric, NDCG, to evaluate models with the cut-off of 20 because 20 recipes are served per page on Cookpad. MAP is also widely used in the literature, but I do not employ it for the reason I mentioned in Chapter 2.3.2.

Regarding loss, Listwise losses are known to be more effective than pairwise losses, but I employ the pairwise cross-entropy loss for two reasons. Firstly, listwise losses are less prevalent in the industry due to its computational cost. Since the outcome of my research is aimed at operating in a production environment, the performance requirements cannot be disregarded. Secondly, I emphasize the reproducibility of my research. Listwise losses require a listwise dataset for training, which would make reproducing the experiments difficult because there are few listwise datasets publicly

Model	Metrics	Loss
HCAN [59]	Acc, macro-F1, MAP, MRR, P@30	Pairwise Cross Entropy
CEDR [47]	ERR@20, NDCG@20, ERR@20	Pairwise Cross Entropy
SNRM [77]	MAP, NDCG@20, P@20, Recall	Pairwise Hinge Loss
Conv-KNRM [18]	NDCG@1, NDCG@10	Pairwise Hinge Loss
NPRF [40]	MAP, NDCG@20, P@20	Pairwise Hinge Loss
NRM-F [78]	NDCG@1, NDCG@10	Pairwise Cross Entropy
DRMM [27]	MAP, NDCG@20, P@20	Pairwise Hinge Loss
DSSM [32]	NDCG@1, NDCG@3, NDCG@20	Pairwise Loss

Table 5.4: Metrics and losses from recent papers

available. On the other hand, pairwise losses can be constructed using pointwise logs, which may be why pairwise losses are prevalent in the literature.

The entire dataset is divided into 10 sets by timestamp to obtain a sufficient number of individual datasets. Each dataset is further divided by timestamp, with the first 75% used for training and the remaining 25% for validation. The effectiveness of models is validated by statistical significance tests.

5.4 RQ1. Is it important to learn feature interactions?

Some papers claimed that interaction-based models generally outperform representation-based models [53, 34]. It is intuitively understandable as representation-based models have its limitation because of the nature that each representation is formed without knowledge of the others, but is this claim valid in recipe search? I first assess the idea to discuss interactions further.

5.4.1 Experimental setup

I compare the following three models: no interaction learning, implicit interaction learning, and explicit interaction learning models.

- **Representation-based model (No feature interaction)** As a no interaction learning model, I employ a simple representation-based model that consists of two component: query encoder and recipe encoder. Both encoders transform entities into vectors and their cosine similarity is computed at the last layer.

Model	NDCG@20
No interaction learning	0.6376
Implicit interaction learning	0.6429
Explicit interaction learning	**0.6483**

Table 5.5: Performance comparison of interaction learning

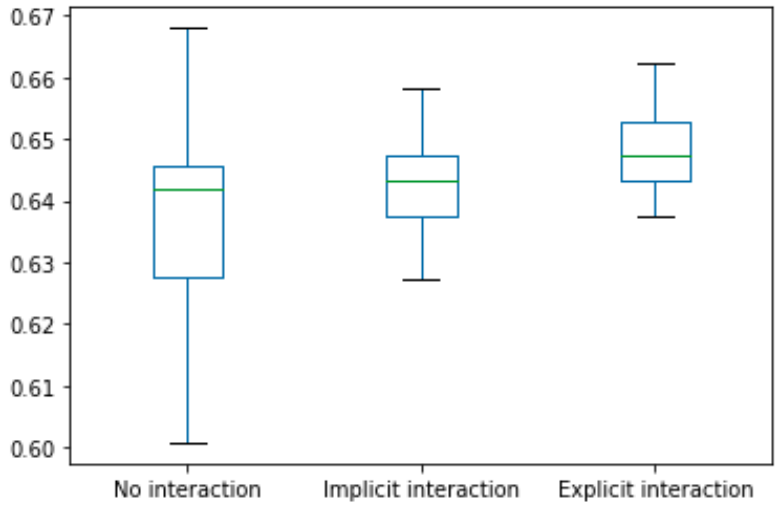


Figure 5.3: Boxplot showing NDCG scores

- **Implicit interaction-based model**: The naive interaction-based model simply concatenates all features at the first layer, then the output is fed to several fully-connected layers.

- **NRM-F-based model)**: This model is based on NRM-F but the text representation is simplified in accordance with the dataset as mentioned in section *5.2*.

5.4.2 Results and Discussion

Table 5.5 shows that the interaction-based models outperformed the representation-based model in average performance. I also performed Tukey's multiple comparison test to see if there is a difference between those models. 5.6 shows that no statistical significance is observed in this experiment. Figure 5.3 is the boxplot showing the performance of each model. The performance of the representation-based model fluctuates.

I examined what was said in the literature: interaction-based models are usually

Pair	p-value
No interaction - Implicit interaction	0.619
No interaction - Explicit interaction	0.159
Implicit interaction - Explicit interaction	0.616

Table 5.6: P-values of Tukey's test on pairs

Model	NDCG@20
NRM-F-based model (query-field interactions)	**0.6483**
NRM-F-based model (all interactions)	0.6403
FM-based model (query-field interactions)	**0.6674**
FM-based model (all interactions)	0.6616

Table 5.7: Performance comparison of models with different interactions

better than representation-based models, but this experiment was not the case. In the first place, this comparison may not make sense because of two reasons. Firstly, it has been proved that feed forward neural networks model low-rank relations [4], meaning that simple implicit interaction learning models can mimic the behavior of any explicit interaction learning models. Secondary. The next experiment shows that FwFM outperforms NRM-F, meaning that different architectures have different performance. The conclusion can vary which model you used as the representative in the experiment.

5.5 RQ2. Is it necessary to limit feature interactions?

Query-field interactions are considered to be important in document ranking, whereas recommendation models do not distinguish between context and item features. It raised the question of why we do not feed all feature interactions as recommendation models do.

5.5.1 Experimental setup

The purpose of this experiment is to investigate whether limiting interactions helps to improve performance. To do so, I train two models with different feature interactions.

- **NRM-F (query-field)**: Consider query-field interactions (as the original implementation).

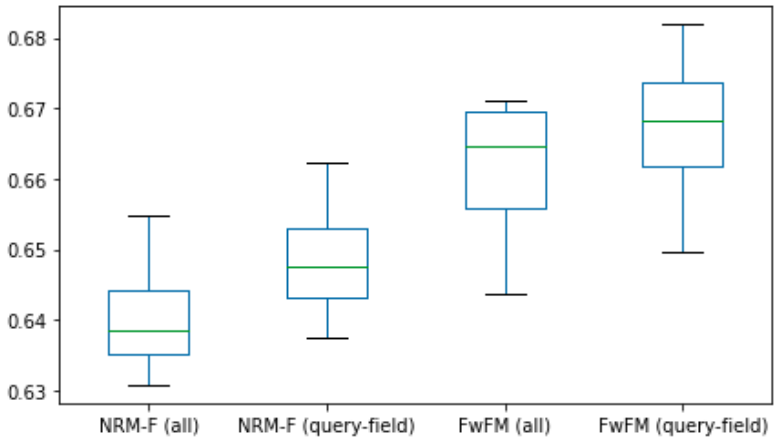


Figure 5.4: Boxplot showing NDCG scores

- **NRM-F (all)**: Consider all feature interactions without distinguishing between query and fields.
- **FwFM (all)** : Consider all feature interactions without distinguishing between query and fields (as the original implementation).
- **FwFM (query-field)**: Consider query-field interactions.

5.5.2 Results and Discussion

Table 5.7 shows the performance of the above mentioned models. Interestingly, models that learned query-field interactions outperformed models trained using all interactions in both NRM-F and FwFM. Table 5.8 shows their p-values are low enough, meaning that it happens not by chance.

This may be because query-field interactions are particularly important in document ranking, and adding irrelevant feature interactions may introduce noise into the model, resulting in degraded performance. I further discuss this topic in *RQ4*.

The experiment also has shown that recommendation models could be used for ranking tasks as it is since FwFM outperformed the simplified NRM-F. Besides, we can further improve performance by incorporating the properties of information retrieval into recommendation models.

Pair	p-value
NRM-F (query-field) - NRM-F (all)	0.0031
FwFM (query-field) - FwFM (all)	0.001

Table 5.8: P-values of paired t-tests on each method

5.6 RQ3. Does feeding first-order features contribute to effectiveness?

The original implementation of NRM-F does not use first-order features. This may be because feeding those features are considered to overfit the model. For example, if there is a feature that indirectly represents the item's popularity, it could appear at the top of the list regardless of the user's intent. However, recommendation models usually do not care about the case. In a sense, feeding first-order features could improve performance because machine learning models eventually update parameters to minimize the loss.

5.6.1 Experimental setup

I examine how performance changes when feeding first-order features. The features I use in this experiment are query, title, description, ingredient, and country. Which second-order interactions to use vary by model as described below.

- **NRM-F (2nd)**: Use second-order query-field interactions only (as the original implementation).
- **NRM-F (1st + 2nd)**: Use first-order features along with second-order query-field interactions.
- **FwFM (1st + 2nd)** : Use first- and second-order interactions (as the original implementation).
- **FwFM (2nd)**: Use second-order interactions only.

5.6.2 Result and Discussion

The boxplot 5.5 shows the performance of the models above. It can be seen that there is no difference between NRM-F (2nd) and NRM-F (1st + 2nd) while FwFM (1st +

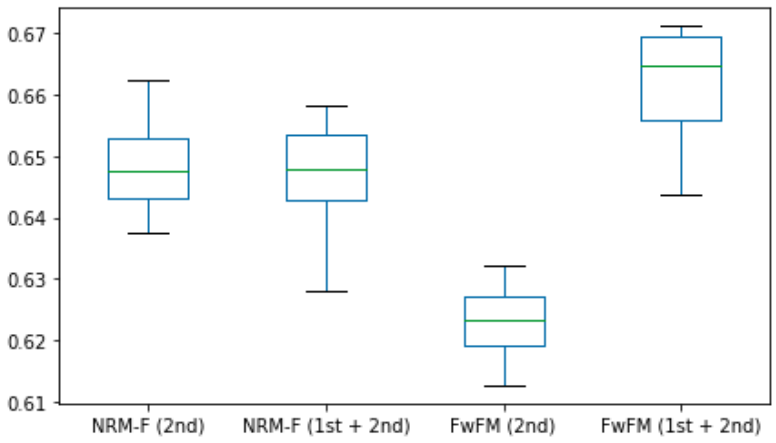


Figure 5.5: Boxplot showing NDCG scores

Pair	p-value
NRM-F (2nd) - NRM-F (1st + 2nd)	0.526
FwFM (2nd) - FwFM (1st + 2nd)	0.0

Table 5.9: P-values of paired t-tests on each method

2nd) significantly outperformed FwFM (2nd) in p-value (Table 5.9). Why only FwFM's performance was improved may be because FwFM is more robust to noisy features to some extent since FwFM has weights that decrease the impact of not important features. The result suggests that feeding first-order features potentially improve effectiveness.

5.7 RQ4. Are there any other important feature interactions besides query-fields?

RQ2 suggests that there are useful and not useful feature interactions. It naturally raises another question: are there any other important feature interactions besides query-field interactions? If such useless feature interactions can be identified, we can further optimize the model.

Model	NDCG@20
FwFM (all)	0.662
FwFM (query-field)	**0.667**
FwFM (selected)	0.665

Table 5.10: Performance comparison of models with different interactions

5.7.1 Experimental setup

I used the FwFM trained using first- and second-order interactions (5 features + 10 feature interactions in total), which is the same as in the experiment in RQ3. FMs compute the scores for each feature independently and sum them up to produce the final score. I trained the model regularly and extracted the individual feature scores on validation data as shown in the following code.

```
class FwFM(BaseModel):
    def build(self):
        ...
        x = tf.concat([first_order_features, feature_interactions],
    axis=1)
        # It is individually computed scores.
        scores = tf.keras.Model(inputs=inputs, outputs=x)
        # Sum up the computed scores, which will be the final score.
        x = tf.keras.backend.sum(x, axis=1, keepdims=True)
        output = layers.Activation('sigmoid', name='label')(x)
        final_score = tf.keras.Model(inputs=inputs, outputs=output,
    name=self.name)
        # `scores` model is used to extract individual scores.
        return final_score, scores
```

I sort features by the correlation to the label and train another model with selected features based on the assumption that the correlation should be a proxy indicator of feature importance since the sum of individual scores will be the final score. Then, I compare the performance of these three models.

5.7.2 Results and discussion

Figure 5.6 shows the distributions of the activation of features. The label represents 1: clicked and 0: not clicked. The x-axis shows the actual value, and the y-axis is the

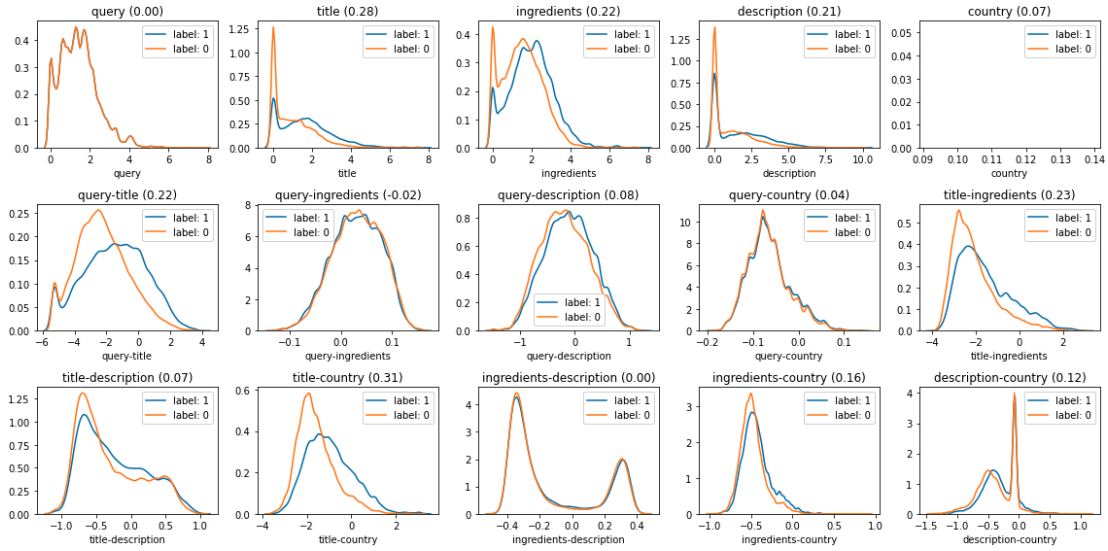


Figure 5.6: Distribution of outputs of the interaction layer

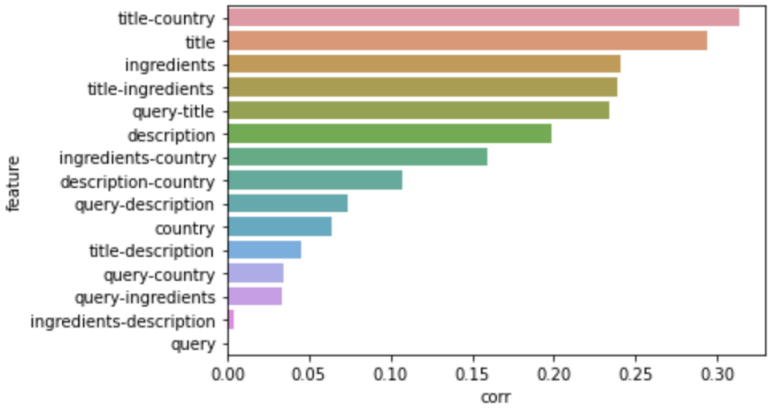


Figure 5.7: The correlation to the label of each feature

probability density. The numbers in parentheses indicate the correlation with the label. Figure 5.7 shows the correlation to the label of each feature sorted by its value.

It makes sense that the shape of the distribution of query-title is more different than other features. Furthermore, the orange line of the query exactly matches that blue line; their correlation is zero. It suggests that the model cannot guess which recipe is more likely to engage by just looking at a query. It also makes sense. From these, we may say that the correlation represent the importance of the features to some extent.

However, table 5.10 shows the performance of the FwFMs with different features. Though the interpretation of the correlation was intuitive, the model with selected features did not outperform the mdoel with query-field interactions.

Chapter 6

Conclusion

6.1 Summary of outcomes

In this project, I have discussed the design of ranking models for multiple field documents. I first reviewed the literature of information retrieval to summarize how researchers have attempted to capture text significance in the classical task setting. Next, I introduced how the community of recommender systems model structured data for the ranking task. Then, I discussed the similarity between these two communities and how to combine both techniques. In my experiments, I implemented variants of NRM-F and FwFM and observed how changes in architecture affect effectiveness. The validity of experiments is an issue that has recently received attention in both communities. I have published the code on GitHub so that other researchers can reproduce my experiments.

6.2 Summary of contributions

I summarize the key contributions of this project as follows.

- **Question on the claim about architectural differences**: Despite a common belief that interaction-based models usually outperform representation-based models, my experiment has shown that depending on the choice of the implementation, there is no statistical difference in our dataset (*RQ1*).
- **The importance of selecting important feature interactions**: My experiment

has shown that models in recommender systems potentially show better performance. In addition, incorporating the characteristics of document ranking into recommendation models can further improve effectiveness. Especially, it has become clear that query-field interactions are the key to effectiveness in the ranking task (*RQ2*).

- **The effectiveness of feeding first-order feature interactions**: Feeding first-order features thought to overfit ranking models, but my experiment has shown that it is not always correct (*RQ3*).
- **Measuring the feature importance of FMs**: To understand which feature is considered important, I computed the activation of each feature and plotted them with the correlation to the label. The correlation seems to represent its feature importance to some extent but the reliability of this method needs to be improved (*RQ4*).

6.3 Areas for improvement and future work

The following are some topics I could not cover in this project, and avenues for future work that could improve or extend the outcomes.

- **Automatic feature interaction learning**: In this project, I focused on feature interactions and examined each feature importance. However, the trend in the community of recommender systems is automatically selecting feature importances in the architecture instead of modifying the architecture in accordance with the measured importance by hand. It can be hypothesized that the methods recently proposed for recommendation tasks also work in the re-ranking task setting, as my experiment has shown.
- **Better term matching**: In multi-field document ranking, queries could contain multiple terms that are supposed to look up different fields. My experiment focused on query-level matching, but term-level matching could better capture patterns of relevance. For instance, the terms in the query ”affordable fridge” are probably supposed to match $\mathrm{affordable} \times \mathrm{price}$ and $\mathrm{fridge} \times \mathrm{product\ title}$ respectively.

- **Deeper understanding of the behavior of models**: I discussed why FwFM outperformed NRM-F, but I could not theoretically prove it. More experiments or a theoretically established method would require to understand models' behavior to advance this field further.
- **Evaluating models beyond accuracy**: There are many different types of search applications, and their requirements vary like some applications want to respond to natural language queries, some applications are more concerned with lexical matching, etc. As we have seen, the architecture of a model affects its behavior as well as its accuracy, but search systems have been evaluated on a single aspect. You do not know if that approach in the literature works for your system. If we can quantify its strength from multiple aspects, we will be able to select the best model for the system.

Bibliography

[1] State-of-the-art table for ad-hoc information retrieval on trec robust04.

[2] Leif Azzopardi, Ryen W White, Paul Thomas, and Nick Craswell. Data-driven evaluation metrics for heterogeneous search engine result pages. In *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*, pages 213–222, 2020.

[3] Nicholas J Belkin and W Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992.

[4] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54, 2018.

[5] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 75–78, 2019.

[6] Sebastian Bruch, Xuanhui Wang, Mike Bendersky, and Marc Najork. An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR 2019)*, 2019.

[7] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[8] Christopher J Burges, Robert Ragno, and Quoc V Le. Learning to rank with non-smooth cost functions. In *Advances in neural information processing systems*, pages 193–200, 2007.

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

[10] Stefan Büttcher, Charles LA Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. Mit Press, 2016.

[11] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[12] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009.

[13] Daniel Cohen, Qingyao Ai, and W Bruce Croft. Adaptability of neural networks on varying granularity ir tasks. *arXiv preprint arXiv:1606.07565*, 2016.

[14] David Cossock and Tong Zhang. Subset ranking using regression. In *International Conference on Computational Learning Theory*, pages 605–619. Springer, 2006.

[15] Alberto Costa and Fabio Roda. Recommender systems by means of information retrieval. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, pages 1–5, 2011.

[16] Koby Crammer and Yoram Singer. Pranking with ranking. In *Advances in neural information processing systems*, pages 641–647, 2002.

[17] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. *CoRR*, abs/1907.06902, 2019.

[18] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 126–134. ACM, 2018.

[19] Hercules Dalianis. Evaluating a spelling support in a search engine. In *International Conference on Application of Natural Language to Information Systems*, pages 183–190. Springer, 2002.

[20] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[21] 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.

[22] Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*, 2017.

[23] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

[24] Norbert Fuhr. Probabilistic models in information retrieval. *The computer journal*, 35(3):243–255, 1992.

[25] Norbert Fuhr. Proof by experimentation? towards better ir research. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2–2, 2020.

[26] Fredric C Gey. Inferring probability of relevance using the method of logistic regression. In *SIGIR'94*, pages 222–231. Springer, 1994.

[27] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. *CoRR*, abs/1711.08611, 2017.

[28] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. Applying deep learning to airbnb search. *CoRR*, abs/1810.09591, 2018.

[29] William Hersh, Andrew Turpin, Susan Price, Benjamin Chan, Dale Kramer, Lynetta Sacherek, and Daniel Olson. Do batch and user evaluations give the same results? In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 17–24, 2000.

[30] Fuxing Hong, Dongbo Huang, and Ge Chen. Interaction-aware factorization machines for recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3804–3811, 2019.

[31] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.

[32] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338. ACM, 2013.

[33] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. A position-aware deep model for relevance matching in information retrieval. *CoRR*, abs/1704.03940, 2017.

[34] Shiyu Ji, Jinjin Shao, and Tao Yang. Efficient interaction-based neural ranking with locality sensitive hashing. In *The World Wide Web Conference*, pages 2858–2864, 2019.

[35] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[36] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[37] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50, 2016.

[38] Jin Young Kim and W Bruce Croft. A field relevance model for structured document retrieval. In *European Conference on Information Retrieval*, pages 97–108. Springer, 2012.

[39] Yanyan Lan, Yadong Zhu, Jiafeng Guo, Shuzi Niu, and Xueqi Cheng. Position-aware listmle: A sequential learning process for ranking. In *UAI*, pages 449–458, 2014.

[40] Canjia Li, Yingfei Sun, Ben He, Le Wang, Kai Hui, Andrew Yates, Le Sun, and Jungang Xu. NPRF: A neural pseudo relevance feedback framework for ad-hoc information retrieval. *CoRR*, abs/1810.12936, 2018.

[41] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

[42] Heran Lin, Pengcheng Xiong, Danqing Zhang, Fan Yang, Ryoichi Kato, Mukul Kumar, William Headden, and Bing Yin. Light feed-forward networks for shard selection in large-scale product search.

[43] Jimmy Lin. The neural hype and comparisons against weak baselines. In *ACM SIGIR Forum*, volume 52, pages 40–51. ACM New York, NY, USA, 2019.

[44] Weiwen Liu, Ruiming Tang, Jiajin Li, Jinkai Yu, Huifeng Guo, Xiuqiang He, and Shengyu Zhang. Field-aware probabilistic embedding neural network for ctr prediction. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 412–416, 2018.

[45] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

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

[47] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. Cedr: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104. ACM, 2019.

[48] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Variational bayesian context-aware representation for grocery recommendation. *arXiv preprint arXiv:1909.07705*, 2019.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[50] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*, 2016.

[51] Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, and Emine Yilmaz. Incorporating query term independence assumption for efficient retrieval and ranking using deep neural networks. *arXiv preprint arXiv:1907.03693*, 2019.

[52] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: a human-generated machine reading comprehension dataset. 2016.

[53] Yifan Nie, Yanling Li, and Jian-Yun Nie. Empirical study of multi-level convolution models for ir based on representations and interactions. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 59–66, 2018.

[54] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. Semantic product search. *CoRR*, abs/1907.00937, 2019.

[55] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *CoRR*, abs/1901.04085, 2019.

[56] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 1349–1357, 2018.

[57] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[58] Benjamin Piwowarski and Patrick Gallinari. A machine learning model for information retrieval with structured documents. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 425–438. Springer, 2003.

[59] Jinfeng Rao, Linqing Liu, Yi Tay, Wei Yang, Peng Shi, and Jimmy Lin. Bridging the gap between relevance matching and semantic matching for short text similarity modeling. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5373–5384, 2019.

[60] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.

[61] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, 2004.

[62] Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

[63] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*, pages 232–241. Springer, 1994.

[64] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[65] Mark Sanderson. *Test collection based evaluation of information retrieval systems*. Now Publishers Inc, 2010.

[66] Fatemeh Sarvi, Nikos Voskarides, Lois Mooiman, Sebastian Schelter, and Maarten de Rijke. A comparison of supervised learning to match methods for product search. *arXiv preprint arXiv:2007.10296*, 2020.

[67] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Advances in neural information processing systems*, pages 961–968, 2003.

[68] Krysta M Svore and Christopher JC Burges. A machine learning approach for improved bm25 retrieval. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1811–1814, 2009.

[69] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 77–86, 2008.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[71] Ross Wilkinson. Effective retrieval of structured documents. In *SIGIR'94*, pages 311–317. Springer, 1994.

[72] SK Michael Wong and YY Yao. Linear structure in information retrieval. In *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 219–232, 1988.

[73] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, 2007.

[74] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using lucene. *Journal of Data and Information Quality (JDIQ)*, 10(4):1–20, 2018.

[75] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. Critically examining the" neural hype" weak baselines and the additivity of effectiveness gains from neural ranking models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1129–1132, 2019.

[76] Hamed Zamani and W. Bruce Croft. Joint modeling and optimization of search and recommendation. *CoRR*, abs/1807.05631, 2018.

[77] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 497–506, New York, NY, USA, 2018. ACM.

[78] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. Neural ranking models with multiple document fields. *CoRR*, abs/1711.09174, 2017.

[79] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. Recommending what video to watch next: A multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, pages 43–51, New York, NY, USA, 2019. ACM.