Study on Feature Interactions in Multiple Field Document Ranking (WIP)

Abstract

We are surrounded by various types of 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 for modern search applications, despite the fact that the quality of search has a direct impact on revenue for companies.

In this thesis, I discuss the design of document ranking models from a new perspective: the relationship between re-ranking tasks and recommendation tasks. I hypothesized that the top-k re-ranking task in industrial applications are closer to recommendation rather than traditional document ranking in the sense that the goal of the task is to find the optimal permutation of complex structured items for a given context. It is based on the assumption that in the top-k re-ranking task, irrelevant documents are already filtered out to some extent and therefore lexical features are not as important as one might think.

I first compare the performance of models that learned feature interactions in different ways. Next, I measure and visualize the importance of feature interactions in order to address the research questions. My experiments have shown that there are many similarities between document re-ranking and recommendation, but it also suggests that text-matching signals still play an important role in re-ranking tasks. In addition, it has been observed that the strength of textual features vary in accordance with the search space.

Contents

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 on a daily basis. The quality of such search applications is crucial not only for users but also for companies because it has a direct impact on revenue. Many companies have been making great efforts to improve the performance of their search systems.

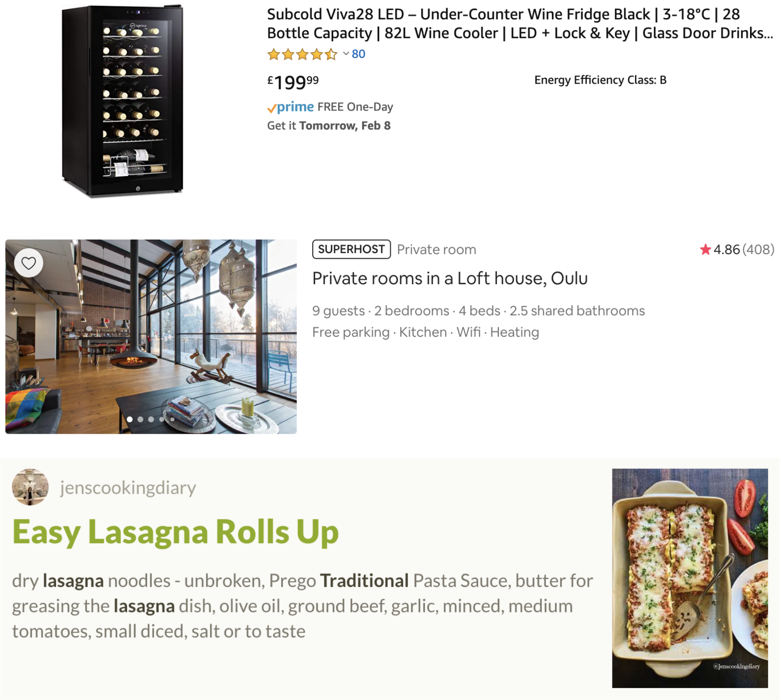
The information retrieval community has mostly focused on pure document retrieval, in which given a query, estimate which text is more relevant. This experimental setting is still mainstream in today's literature. However, with the development of the internet, the forms of search applications are becoming more complex to address growing diverse demands. Figure 1.1 shows example documents from web services (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, modern documents consist of various fields such as title, description, image, review score, facilities, etc, in which it is difficult to capture text significance from short text fields, traditional methods for text cannot be applied to non-text fields, and it is unclear how to combine scores of different fields. It has resulted that traditional methods based on the assumptions made in the 1980s may not be practical any longer. As a consequence, practical search applications often

[1] `https://amazon.com`

[2] `https://airbnb.com`

[3] `https://cookpad.com`



Subcold Viva28 LED – Under-Counter Wine Fridge Black | 3-18°C | 28 Bottle Capacity | 82L Wine Cooler | LED + Lock & Key | Glass Door Drinks...

80

£199^{99}

Energy Efficiency Class: B

prime FREE One-Day

Get it **Tomorrow, Feb 8**

SUPERHOST Private room

★4.86 (408)

Private rooms in a Loft house, Oulu

9 guests · 2 bedrooms · 4 beds · 2.5 shared bathrooms

Free parking · Kitchen · Wifi · Heating

jenscookingdiary

Easy Lasagna Rolls Up

dry **lasagna** noodles - unbroken, Prego **Traditional** Pasta Sauce, butter for greasing the **lasagna** dish, olive oil, ground beef, garlic, minced, medium tomatoes, small diced, salt or to taste

Figure 1.1: Examples from

employ heuristic methods to supplement information signals that are not considered. This is suboptimal and the system would suffer from endless empirical performance tuning.

To the best of my knowledge, NRM-F proposed by Zamani et al. [63] is the only paper that discusses ranking models for semi-structured multi-field documents from an architectural perspective. The model maps each field to a latent space and generates query-field vectors. These vectors are then concatenated and fed into fully-connected layers to produce a final score. It outperformed existing methods but some questions were raised such as query-field vectors are simply concatenated but the model disregards the fact that the strength of association between features may be different.

Multiple field document ranking is still in its infancy and what is ultimately effective may depend on the application. Therefore, a review on methods from a holistic perspective is needed. In this thesis, I shed light on interactions between fields, and demystify what ranking models essentially learn 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.

RQ1. Is limiting feature interaction effective? NRM-F explicitly learns query-field 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. How important are the text matching signals? While NRM-F focuses on query-field interactions, there must be other important feature interactions to consider. I measure and visualize feature interactions to locate such feature interactions. Further more, I investigate how the importance of feature interactions vary by search space.

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 1960s, which was 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 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. In this chapter, I describe 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. Regarding neural networks, I use [.; .] as a concatenation operation. For example,

$$\mathrm{x} = [\mathrm{x}_1; \mathrm{x}_2; \ldots; \mathrm{x}_\mathrm{N}]$$

shows that x is a concatenation of N vectors. In this thesis, I use terms *documents* and

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
T	Full vocabulary of all terms
$s_{q,d}$	Relevance score against a pair of query q and document d
R_q	Set of ranked results retrieved for query q
$\langle i, d \rangle$, where $\langle i, d \rangle \in R_q$	Result tuple (document d at rank i)
$rel_q(d)$	Degree of relevance of document d to query q
$rel_q\left(d_i\right) > rel_q\left(d_j\right)$	d_i is more relevant than d_j for query q
$df(t)$	Number of documents that contain term t
$\vec{v}_z$	Vector representation of text z
$p(\mathcal{E})$	Probability function for an event $\mathcal{E}$

Table 2.1: Notation used in this thesis

items interchangeably.

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, and 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 has been a problem discussed long time in the information retrieval community [53]. Broadly speaking, evaluation metrics can be categorized into two types: online metrics or offline metrics. Online metrics are generally measured using search logs. They are usually more user-focused than offline metrics. Examples of online metrics include Session Abandonment Rate, Click-Through Rate, Session Success Rate, and Zero Result Rate. Offline metrics are calculated by search session and then average them. Precision, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG) are in this category.

Those offline evaluation on the performance of ranking models is carried out by comparison between the ranking lists output by the model and the ranking lists given as the ground truth. Ranking utilities such as MAP and NDCG are designed to emphasize the top of the ranked list in a way that mimics user behavior that users are less likely to examine documents at larger rank positions, thus the contribution to the utility should decay as its rank increases.

Mean Average Precision (MAP)

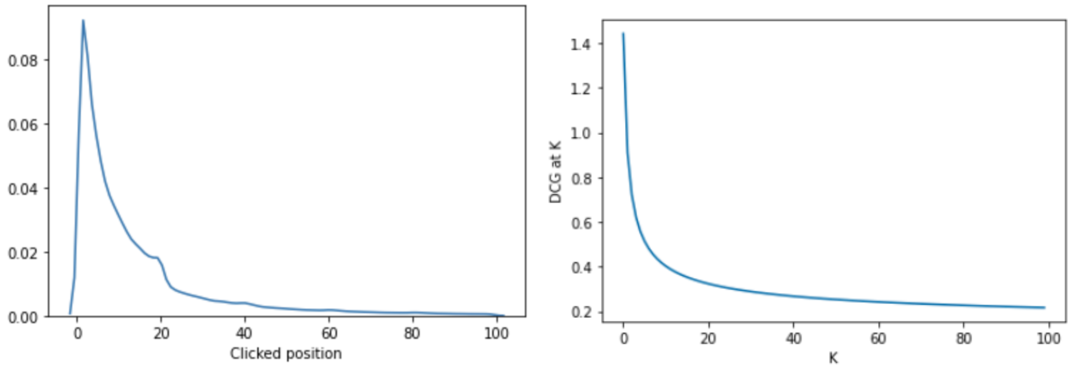
Given Q queries, Precision at cut-off k is computed as follows.

$$\text{Precision}\ @k = \sum_{i=1}^{k} rel_q\left(d_i\right)$$

where d_i is the i^{th}-ranked document returned by the system in response to query q, and $rel(d_i)$ is the binary relevance assessment of this document. Then we can compute Average Precision (AP) as follows.

$$\text{MAP} = \frac{\sum_{i=1}^{k} rel_q\left(d_i\right) \cdot \text{Precision}\ @k}{\#\{\text{relevant documents}\}}$$

MAP is the average of AP over all queries.



(a) The distribution of clicked positions

(b) DCG at position K

Figure 2.1: The distribution of clicked positions and gain

Normalized Discounted Cumulative Gain (NDCG)

Discounted Cumulative Gain (DCG) measures the goodness of the ranking list with the labels. Specifically, DCG at position k is defined as

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

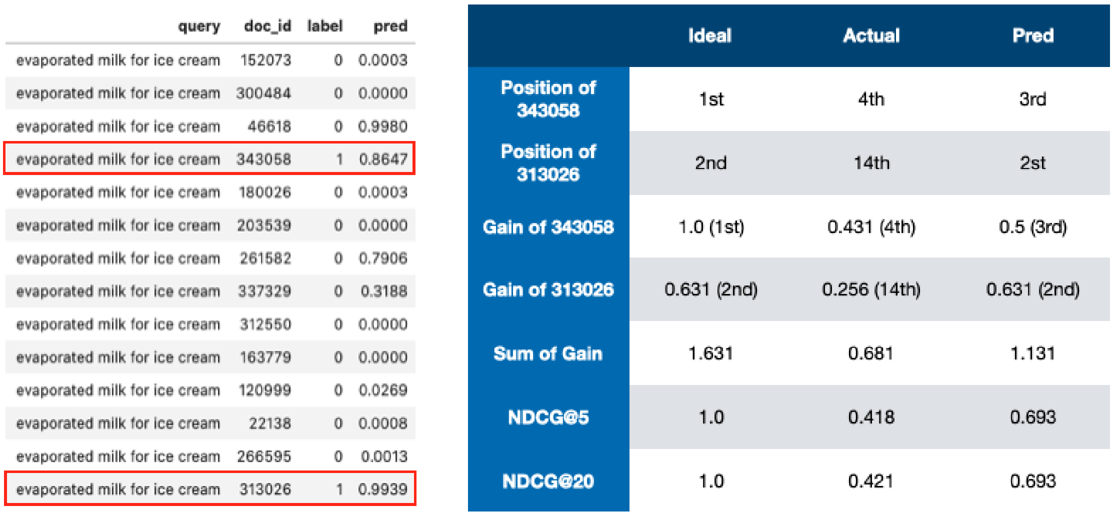
In binary relevance judgement, the numerator simply becomes 0 or 1. The gain at position k decays in a logarithmic fashion like below.

$$\text{DCG} = rel(d_2) + \frac{rel(d_2)}{\log_2 2} + \frac{rel(d_3)}{\log_2 3} + \ldots \frac{rel(d_k)}{\log_2 k}$$

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

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

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 suitable metric for the application. Note that a drop in the distribution of actual clicked position is because 20 documents are retrieved per



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

page.

Figure 2.2 shows an example of how NDCG is calculated. In this search session, two documents are clicked. The ideal list have 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 position, 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.

Though these metrics are widely used in the literature, the reliability and reproducibility of experiments have been criticized [17].

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 in accordance with 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 employing 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 make a guess of whether a document satisfies the query under an uncertain understanding of the user query.

The idea of ranked retrieval was proposed by Luhn [35], and Maron, Kuhns, and Ray tested the idea that assigns a score that indicates the relevance of each document to a given query, and 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 importance of the keywords to the documents using a probabilistic approach. In the same year, Luhn suggested that word frequency and distribution in a document represents a useful measurement of word significance [36]. This approach later became known as term frequency weighting.

Sparck Jones complemented Luhn's term frequency (TF) weights [29]. This paper introduced the idea of inverse document frequency (IDF) that the frequency of occurrence of a word in a document collection was inversely proportional to its significance. Then, combining these two weights (TF-IDF) was quickly adopted.

In 1977, Stephen Robertson defined the probability ranking principle [50] stating that the overall effectiveness of the information retrieval system will be maximized if the system's response to each query is a ranking of the documents in the collection in the order of decreasing probability of relevance. Since the original probabilistic model did not take into consideration term frequency weights, a number of methods were proposed to incorporate term frequency in an effective way. The variations of TF-IDF weighting schemes were proposed in this era. This work led to the ranking function BM25 [51], 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 stating that the overall effectiveness of the information retrieval system will be maximized if the system's response to each query is a ranking of the documents in the collection in the order of decreasing probability of relevance [8]. A question arises here: what is the probability that this document is relevant to this 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.

$$\begin{align} log\frac{p(rel|d, q)}{1 - p(rel|d, q)} &= log\frac{p(d, q|rel)p(rel)}{p(d, q|\overline{rel})p(\overline{rel})} \tag{2.1} \\ &= log\frac{p(d|q, rel)}{p(d|q, \overline{rel})} + log\frac{p(rel|q)}{p(\overline{rel}|q)} \tag{2.2} \end{align}$$

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, one of the popular instantiations 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 drawn. 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 considered 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 Vector Space Model in 1975 [52]. The idea is to represent both queries and documents as vectors in a latent Euclidean space and rank documents according to similarity to the query. The standard way of quantifying the similarity between two vectors is to compute the cosine similarity of their vector representations.

$$f(q,d) = \frac{\overrightarrow{V}(q)\cdot\overrightarrow{V}(d)}{\left|\overrightarrow{V}(q)\right|\left|\overrightarrow{V}(d)\right|}$$

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 $\overrightarrow{x}\cdot\overrightarrow{y}$ of two vectors is defined as $\sum_i x_i y_i$. Euclidean length is defined as $\sqrt{\sum_i \overrightarrow{V}_i}$. Thus, it can be viewed as the dot product of the normalized versions of the two vectors.

The most well-known method is Latent Semantic Indexing (LSI), where the dimensionality of the vector space of a document collection was reduced through singularvalue decomposition [15]. More research on learning representations of queries and documents has been done after the breakthrough of neural networks. This method later became called representation-based approach.

In practice, computing the cosine similarities between the query vector and each document in the set, sorting the resulting scores and selecting the top k documents can be expensive. For that reason, 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 really want is not reflected to 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

Learning to rank is a task to automatically learn a ranking model using training data, such that the model can sort new documents according to their degrees of relevance. Learning to rank uses relevance labels, typically click data as implicit feedback, are used to train models. This is distinct from the models in the previous chapter.

This new field began to be explored in the late 1980s [60, 19, 20], but those attempts were not very successful until 2000s. This is because traditional ranking functions in information retrieval used a very small number of features such as term frequency, inverse document frequency, document length. It was possible to tune weights by hand. In addition, the amount of data that can be used for training was limited, meaning that it was difficult 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 basically a supervised learning task and thus has training and testing phrases. Most learning to rank methods learn the ranking functions by minimizing loss functions. The process of learning to rank is as follows. In training, a number of sets are given, each set 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 a ranking function is constructed by minimizing a certain loss function on the training data. In testing, given a new set of features, the ranking function is applied to produce a ranked list of the features. Depending on the design of the loss function, 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 document at a time in the loss function and train a model on it to predict the relevance degree of the document. Such a function is usually called scoring function. Based on the scoring function, the system sorts documents and produce the final ranked list. The loss function examines the accurate prediction of the ground truth label for every single document. In this approach, the ranking task is simplified and transformed to classification, regression or ordinal classification. Existing methods for these tasks can be applied. The pointwise approach includes Subset Ranking [12], McRank [33], Prank [13], and SVM for Ordinal Classification [55]. 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 tuned throughout the learning process, but thought to be less effective in ranking tasks because pointwise loss functions do not consider the inter-dependency between documents, and thus the position of a document in the final ranked list is invisible to its loss function. Therefore, better ranking paradigms that directly optimize document ranking 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, and hence, the pairwise approach does not focus on accurately predicting the relevance degree of each document unlike the previous approach. In the sense, it is closer to the concept of ”ranking” than the pointwise approach. In the pairwise approach, ranking is usually reduced to a classification on document pairs, i.e., to determine which document in a pair is preferred. That is, the goal of learning is to minimize the number of inversions in ranked documents. The examples include Ranking SVM [28], RankBoost [18], RankNet [5], LambdaRank [6], and LambdaMART [7]. Pairwise losses generally have the following form.

$$\mathcal{L} = \phi\left(s_i - s_j\right)$$

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 cross entropy theory that is defined n the differences in scores of pairs of documents, defined as follows.

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

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

The pairwise approach is more ideal than the pointwise approach with respect to the ranking objective. However, only the pair document dependence is just considered in pairwise approaches, which means that dependence between each document in the whole rank cannot be fully considered.

3.3 Listwise approach

The idea of listwise ranking objectives is to construct loss functions that directly reflect the performance of the model in ranking. Instead of comparing two documents each time, listwise loss functions compute ranking loss with each query and their candidate document list together. The group structure of ranking is maintained and ranking evaluation measure can be more directly incorporated into the loss functions in learning.

There are two main sub-techniques for listwise learning to rank: direct optimization of measures of information retrieval such as SoftRank [57] and AdaRank [61], and minimizing a loss function that is defined based on understanding the unique properties of the ranking such as ListNet [9] and ListMLE [32].

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 really lead to the optimization of the ranking measures.

Actually, researchers have tried to answer this question, and it has been proved in [33, 10, 3] that the regression and classification based losses used in the pointwise approach are upper bounds of (1-NDCG).

Though it is known that listwise approaches are more desirable for learning to rank because of the nature of the design as shown in [4], pointwise and pairwise approaches tend to be preferred for performance reasons in practice [64].

Chapter 4

Neural Information Retrieval

Neural networks recently have demonstrated impressive performance in various machine learning tasks including information retrieval. State-of-the-art records have been continuously broken by neural ranking models [1, 41]. 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 [26], 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. In this section, I introduce several assumptions and approaches for text representation.

Documents and queries are a sequence of terms. In order 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 dimension corresponded to a term. This method did not take into account word order, so the relationship between words was disregarded. For example, ”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 for the queries ”hot dog” and ”dog hot”, the same documents are retrieved in exactly the same order. Conventional models are based on this assumption, and the effectiveness of BM25 has shown in the history of information retrieval, but how plausible is 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 also has been proved that methods relying on term frequency are not effective for short text retrieval because the distribution of term frequency becomes flat, making capturing word significance difficult.

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 [38].

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 network (LSTM) was the most commonly used network architecture to capture inter-term dependencies. Convolutional neural networks 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 [58] 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 performance of information retrieval systems. Most recently, Mitra et el. have shown that no significant loss is observed in the models that incorporating the query term independence assumption in web search [40]. Another paper has reported that state-of-the-art BERT-based models is mediocre in product search [54]. These may suggest that capturing inter-term dependencies is not critical in some forms of search applications, whereas it is a central concern in question answering.

4.2 Types of ranking models

According to the existing literature [26], the core problem in ad-hoc retrieval can be formalized as a text-matching problem as follows: Given two texts T_1 and T_2, the degree of matching is typically measured as a score produced by a scoring function based on the representation of each text like below.

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

where Φ is a function to map each text to a representation vector, and F is the scoring function based on the interactions between them. Depending on how you choose the two functions, existing neural matching models can be categorized into two types of architectures, namely representation-based model and interaction-based model. Besides these two categories, some models adopt a hybrid approach to obtain the merits of both architectures in learning relevance features.

4.2.1 Representation-based model

The underlying assumption of this type of model is that relevance depends on compositional meaning of the input texts. Therefore, models in this type usually define complex representation function but no interaction function to obtain high-level representations of the inputs.

Representation-based models 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. This approach is based on two assumptions that 1. queries and documents have 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 learned from data. Examples in this regard include DSSM [26], ARC-I [25], and DESM [39].

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 to reduce the dimensionality of feature vectors. The final layer computes the cosine ismilarity between the computed vectors as a measure of their relevance.

ARC-I leverages pre-trained word embeddings and several layers of convolutions and max-pooling to generate separate dense representations for queries and documents. Finally, it applies an MLP to compare the resulting vectors via a non-linear similarity function. ARC-I stacks 1D convolutional layers and max pooling layers are applied on the input texts to produce their high-level representations respectively. ARC-I then concatenates the two representations and applies an MLP as the evaluation function.

4.2.2 Interaction-based model

The underlying assumption of this type of architecture is that relevance is in essence about the relation between the input texts, so it would be more effective to directly learn from interactions rather than from individual representations.

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 [25], DRMM [21], PACRR [27], 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.

Interaction-based models have its limitation because of the nature that each representation is formed without knowledge of the others.

ARC-II is designed to capture interactions between query and document. It 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 similarity between every n-gram pair. After that, several convolutional and max-pooling layers are leveraged to obtain the final relevance score.

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

4.2.3 Hybrid model

In order to take advantage of both representation-based and interaction-based models, a natural way is to adopt a hybrid architecture for feature learning.

Relevance matching models such as DRMM and KNRM resemble traditional information retrieval measures in that they directly consider the similarity of the document with respect to the query. On the other hand, many NLP problems, such as question answering and textual similarity measurement, require more semantic understanding and contextual reasoning rather than specific term matches. In this context, applying contextualized language models to information retrieval has been studied. [43] employed pre-trained language models such as ELMo [45] and BERT [16], that have achieved impressive results on various natural language benchmarks, showed that this approach potentially works in information retrieval tasks as well.

Especially in information retrieval, lexical matching is believed to be important along with semantic matching while question answering does not place importance much on lexical features because questions and answers are heterogeneous, meaning there is an asymmetry between queries and documents in terms of length and richness. In such tasks, textual similarity may not be an appropriate measure to represent relevance. Since such search applications have to consider those different signals, hybrid methods have been proposed such as DUET [21] and HCAN [47]. DUET combines signals from a local model for relevance matching and a distributed model for semantic matching. HCAN also minimizes the losses of both relevance and semantic modules jointly.

DUET combines the strenghs of representation and interaction-based models. DUET calculates the relevance between a query and a document using local and distributed representaions and for this reason, has been classified as a hybrid model.

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). There are a few studies to discuss 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 [59].

In non-neural models, Robertson et al. proposed BM25F [49], an extension to the original BM25 model, arguing that the linear combination of field-level scores is dangerous because it bypasses the careful balance across query terms in the BM25 model. The BM25F solution is to first 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 [46]. Svore and Burges proposed a supervised approach, called LambdaBM25 [56], that is 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 [31].

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 [63]. They have found that learning separate latent spaces for matching a query against different document fields is more effective than using a shared latent space for all fields. This may be because the different document fields may contain information relevant to different aspects of the query intent. They also state that it is better for the ranker to 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.

In NRM-F, both query text and text fields are represented using a character n-gram hashing vector, introduced by Huang et al. [26]. Then, a convolution layer is employed to capture the dependency between terms. However, Amazon's experiment showed that taking the average of term vectors performed similar or slightly better than recurrent units with significantly less training time [42]. That may be because queries are usually given as a set of keywords in product search.

4.4 Factorization machines

While only a few studies on dealing with multiple fields have been done in information retrieval, it is a much more popular topic in the recommendation community.

Most state-of-the-art models in recommendation are based on the factorization machine proposed in 2010 [48]. The factorization machine model the interaction between two features i and j as the dot product of their corresponding embedding vectors v_i, v_j as follows.

$$\Phi_{FMs}(w, x) = w_0 + \sum_i w_i x_i + \sum_{i,j} x_i x_j \langle v_i, v_j \rangle$$

In that sense, factorization machines can be viewed as interaction-based ranking models in information retrieval. This basic idea has worked well in recommendation tasks and be continuously improved up until now.

One of the problems of the original factorization machine is neglecting the fact that a feature might behave differently when it interacts with features from different other fields. In 2016, the field-aware factorization machine was proposed [30] to learn such difference explicitly by $n - 1$ embedding vectors for each feature.

$$\Phi_{FFMs}(w, x) = w_0 + \sum_i x_i w_i + \sum_{i,j} x_i x_j \langle v_{i,F(j)}, v_{j,F(i)} \rangle$$

However, field-aware factorization machines can overfit easily because of the large number of parameters. Hence, the field-aware probabilistic embedding neural network [34] was proposed in 2018. Various confidence levels for feature embeddings are incorporated to enhance the robustness and accuracy. Applying the variational autoencoder to recommendation tasks was also proved effective in grocery recommendation [37].

Since an alternative approach to prevent the overfitting issue, the field-weighted factorization machine [44] was proposed in the same year, arguing how to reduce the number of parameters in 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

$$\Phi_{FwFMs}(w, x) = w_0 + \sum_i x_i w_i + \sum_{i,j} x_i x_j \langle v_i, v_j \rangle r_{F(i),F(j)}$$

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

Since the attention mechanism is known to address such issues, the interaction-aware factorization machine [24] has introduced an attention network to learn feature interaction importance, which simultaneously considers field information as auxiliary information, instead of learning explicit weights.

Information retrieval and recommendation essentially share the same goal: helping people to locate the information they need. Their architectures are often similar to some extent. For instance, both search and recommender systems generate candidates and rank them. Zamani et al. has shown that jointly learning both information retrieval model and recommendation model outperforms models trained independently [62], indicating that approaches working in the literature of recommender systems could potentially work in information retrieval tasks as well.

However, to enumerate all cross features may lead to degraded learning performance, since they may be irrelevant or redundant, introduce noise, and increase the difficulty of learning. Hence, only useful and important cross features should be generated, but they are often task-specific.

Neural Factorization Machines (NFMs) [23] NFM performs a non-linear transformation on the latent space of the second-order feature interactions.

Factorization Machine is a widely used supervised learning approach by effectively modeling of feature feature interactions. Despite the successful application of FM and its many deep learning variatns, treating every feature interaction fairly may degrade the performance. For example, the interactions of a useless feature may introduce noises; the importance of a feature may also differ when interacting with different features.

4.5 Examples of industrial application

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

Airbnb shared the success stories of applying machine learning to their search product [22]. 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 in spite of its success in other applications.

Industrial search applications have a common issue of how to fill the gap between user's information needs and the actual query. In Amazon, they conducted a research project focusing on semantic matching using neural networks [42]. 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 [14], 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 [64]. Typical recommendation systems follow a two-stage design with a candidate generation and a ranking because re-rankers are usually computationally expensive. Scalability is extremely important for them because 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 in order to find the optimal ranked list give a set of candidates.

4.6 Criteria in designing ranking model

As discussed earlier, the optimal architecture varies by the characteristics of the application. Cohen et al. examined how granularity affects the performance of the ranking model [11].

- Text vs. Structured
- Re-ranking vs. Ranking Problem
- Domain-specific vs. General Application
- Keyword Query vs. Natural Query
- Homogeneous vs. Heterogeneous

4.7 Conducting experiments

4.7.1 Criticism of reproducibility

4.7.2 Datasets

TREC collections such as Robust [21, 18], ClueWeb [21], GOV2 [33, 34] and Microblog [33], as well as logs such as the AOL log [27] and the Bing Search log [13, 47, 48, 23]. Recently, a new large scale dataset has been released, called the NTCIR WWW Task [49], which is suitable for experiments on neural ranking models.

Because of the purpose of the research, the dataset needs to be about search logs of structured document retrieval. MS MARCO [1] is a dataset that NRM-F was evaluated on. However, building a ranking model for web search is not a common use case in industry.

4.7.3 Metrics and lossess

In order to follow the standard experiment settings in neural information retrieval, I examined which metrics and losses are employed in recent papers (Table 4.1).

[1] MS MARCO `https://microsoft.github.io/msmarco/`

Model	Metrics	Loss
HCAN	Acc, macro-F1, MAP, MRR, P@30	Pairwise Cross Entropy
CEDR	ERR@20, NDCG@20, ERR@20	Pairwise Cross Entropy
SNRM	MAP, NDCG@20, P@20, Recall	Pairwise Hinge Loss
Conv-KNRM	NDCG@1, NDCG@10	Pairwise Hinge Loss
NPRF	MAP, NDCG@20, P@20	Pairwise Hinge Loss
NRM-F	NDCG@1, NDCG@10	Pairwise Cross Entropy
DRMM	MAP, NDCG@20, P@20	Pairwise Hinge Loss
DSSM	NDCG@1, NDCG@3, NDCG@20	Pairwise Loss

Table 4.1: Metrics and losses in papers

Chapter 5

Ranking Model for Semi-Structured 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 the impact of text relevance in multi-field document ranking. In this chapter, I describe my experiments and discuss the results. First, I introduce the basic experimental settings between experiments.

5.1 Dataset

I conduct experiments on a dataset provided by Cookpad. Cookpad is an online recipe community platform. Recipes are published by users and the platform provides search features. Models were trained using the master recipe data and search logs. The size of the dataset is shown in Table 5.1.

#queries	2,733,549
#unique queries	94,993
#unique presented items	301,078
#clicks	2,733,548

Table 5.1: Basic Statistics of the dataset

Feature	Type
recipe_id	Integer
user_id	String
description	String
ingredients	String list
Steps	String list

Table 5.2: The schema of recipe data

Feature	Type
session_id	Integer
country	String
recipe_id	Integer
position	Integer
query	String
fetched_recipe_ids	String

Table 5.3: The schema of search logs

5.1.1 Recipe data

Recipe data contains attributes of recipe itself as shown in Table 5.2. Ingredients and steps are represented as a list. Though steps are lists, it can be concatenated because it is a sequence that describes how to cook the recipe. Publisher information is represented as user ID. There are several options to feed it to the model such as simply embedding IDs, use learned representations obtained from a representation-based model, take the average of recipe embeddings, etc, but in this experiment I do not use the field for simplicity.

5.1.2 Search logs

A search log is an event log that is issued when a user clicks a recipe in the search results. The available attributes are shown in Table 5.3. Each log has a session ID to group logs as a sequence of events. Note that the same session can contain several click events and different queries. *fetched_recipe_id* shows what recipes were retrieved against the query and *position* shows the clicked recipe position in the list.

Table 5.4 shows the percentage of the number of terms in a query. It can be seen that most of queries have less than 5 terms in a query, and each query looks a set of

Number of terms in a query	Percentage
1	21.33%
2	40.04%
3	22.04%
4	10.09%
5	4.75%
6	1.33%
7	0.23%
8	0.15%
9	0.01%
10	0.01%

Table 5.4: Percentage of the number of terms in a query

keywords. Examples of long queries are ”chicken wings potato japanese style” and ”moist and fluffy sponge cake” though a small number of queries look like a question such as ”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 in the list.

5.2 Data processing

First, I split search logs by timestamp to obtain n datasets. Each dataset is also split by timestamp and 75% of the data is used for training and the rest is used for validation because models usually predict unseen future data.

Search logs are aggregated by session ID and query. Each list is cut off by the clicked position based on the assumption that retrieved items 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 and the number of negatives per positive can be specified in which I set 3 in this experiment.

Text embeddings are shared across text fields (query, title, ingredients, steps, and description). To obtain fix-sized vectors from texts, I took the average of vectors because in this task setting, recipe search can be categorized as a *keyword match domain specific structured document re-ranking task.*

5.3 Training and validation

The performance of models is measured using NDCG@20. I set the cut-off of 20 because 20 items are served per page on Cookpad. I adopt the pairwise cross entropy loss. In any models, they are optimized using Adam optimizer with learning rate 0.001. Effectiveness is validated by statistical significance tests.

5.4 Effectiveness of feature interactions

The literature says that learning feature interactions is the key to effectiveness. First I assess this statement to further discuss feature interactions.

5.4.1 Experimental setup

I compare the following three models: no interaction learning, implicit interaction learning, and explicit interaction learning to see the effectiveness of feature interaction learning.

5.4.2 No interaction learning model

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.

5.4.3 Implicit interaction learning model

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

5.4.4 NRM-F-based models

NRM-F formulates the document representation learning function Φ_D as follows:

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

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.

NRM-F considers the interactions between queries 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.

Regarding the way of encoding text, NRM-F transforms text into character n-grams but in my experience, I employed word-level representations based the assumption that word-level representations are more ideal than character-level representations as long as it fits in memory. NRM-F is designed for web search and its vocabulary size is infinitely large to represent everything in this world. On the other hand, the limited size of vocabulary in domain specific search allows us to employ word-level representations. I examined the vocabulary size of datasets used in NRM-F and Cookpad recipe data. They are 3,958,542 and 229,559 respectively. In both cases, the vocabulary size becomes 17,576 if the character-level trigram representation is used.

5.4.5 Results and Discussion

Table 5.6 shows that interaction-based models outperformed the representation-based model. It may suggest that neural models have the ability of learning interactions to some extent, but having a dedicated architecture helps to improve performance. Figure 5.1 plots the distribution of outputs of a representation-based model and interaction-based model. From the figure, it can be seen that the representation-based model splits positives and negatives well while the interaction-based model can output neutral scores as they are cosine similarity.

Model	Parameters	Val loss	NDCG@20
No interaction learning			
Implicit interaction learning	549,345	0.5133	0.6028
Explicit interaction learning	548,321	0.5045	0.6352

Table 5.5: Performance comparison of models

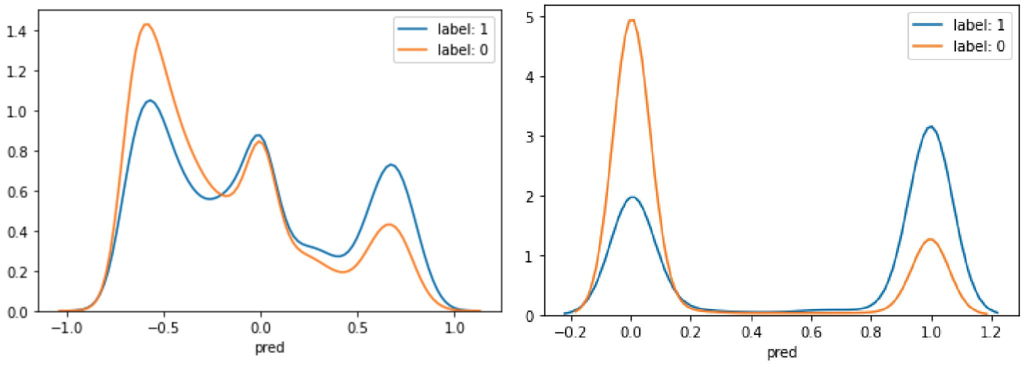


Figure 5.1: Distribution of predictions

5.5 Effectiveness of limiting interactions

Query-field interactions are considered to be important in document ranking. It raised the question of why we do not feed all feature interactions.

5.5.1 Experimental setup

The next experiment compares explicit interaction learning models trained on all feature interactions and trained on limited feature interactions (query-field interactions) to see the effectiveness of limiting interactions. In addition to NRM-F, I add factorization machines that learn all feature interactions to the experiment.

5.5.2 FM-based models

The general form of the factorization machine is as follows.

$$\Phi_{FMs} = w_0 + \sum_i x_i w_i + \sum_{i,j} x_i x_j \langle v_i, v_j \rangle$$

Model	Parameters	Val loss	NDCG@20
NRM-F-based model (query-field interactions)	548,321	0.5045	0.6352
NRM-F-based model (all interactions)	558,561	0.5167	0.6352
FM-based model (query-field interactions)	560,109	0.5054	0.6575
FM-based model (all interactions)	602,073	0.5109	0.6217

Table 5.6: Performance comparison of models

It considers all feature interactions without distinguishing whether or not a feature is a context while NRM-F considers only query-feature interactions defined as follows.

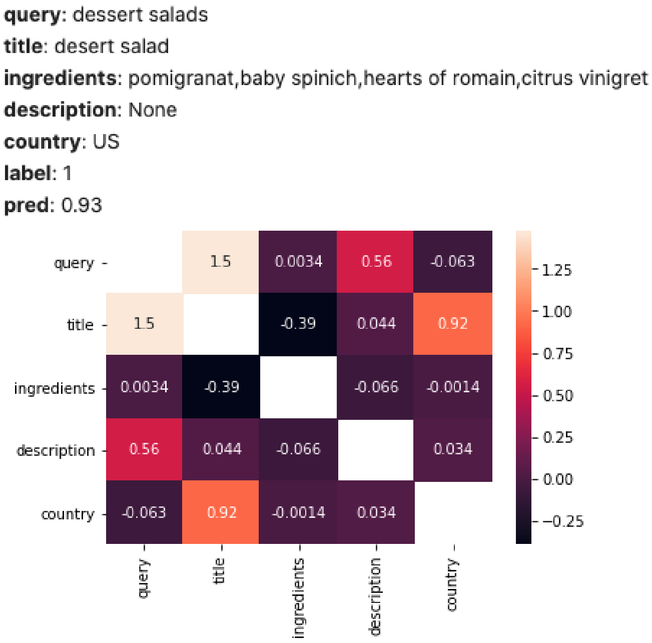
$$\Phi_{NRM-F} = \sum_i \text{query} \odot \text{field}_i \langle v_i \rangle$$

5.5.3 Results and Discussion

Table 5.6 compares the performance of the above mentioned models. Interestingly, models that learned query-field interactions outperformed models trained on all interactions. This may be because considering irrelevant feature interactions could be noise for the model, resulting degraded performance. It means that there are useful and not useful interactions. It naturally raises another question: is there any other important feature interactions besides query-field interactions?

5.6 Strength of feature interactions

”sushi (title) in ”Japan” (country) and *”sushi” (title) in ”UK” (country)* must have different texture, tastes, and characteristics but models that consider query-field interactions disregard the fact. I attempt to measure and visualize feature interactions to locate such feature interactions. Further more, I investigate how the importance of feature interactions vary by search space.



query: dessert salads
title: desert salad
ingredients: pomigranat,baby spinich,hearts of romain,citrus vinigret
description: None
country: US
label: 1
pred: 0.93

	query	title	ingredients	description	country
query		1.5	0.0034	0.56	-0.063
title	1.5		-0.39	0.044	0.92
ingredients	0.0034	-0.39		-0.066	-0.0014
description	0.56	0.044	-0.066		0.034
country	-0.063	0.92	-0.0014	0.034	

Figure 5.2: Activation of interactions of a sample

5.6.1 Experimental setup

I trained a model based on the weighted factorization machine without the global bias and the first order features defined as follows to focus on feature interactions.

$$\Phi = x_i x_j \langle v_i, v_j \rangle r_{F(i),F(j)}$$

To see the strength of associations between features, I plot the distribution of outputs of the interaction layer.

5.6.2 Results and discussion

Figure 5.3 shows the distribution of outputs of the interaction layer. Figure 5.4 shows how the strength of lexical matching signal changes by the search space.

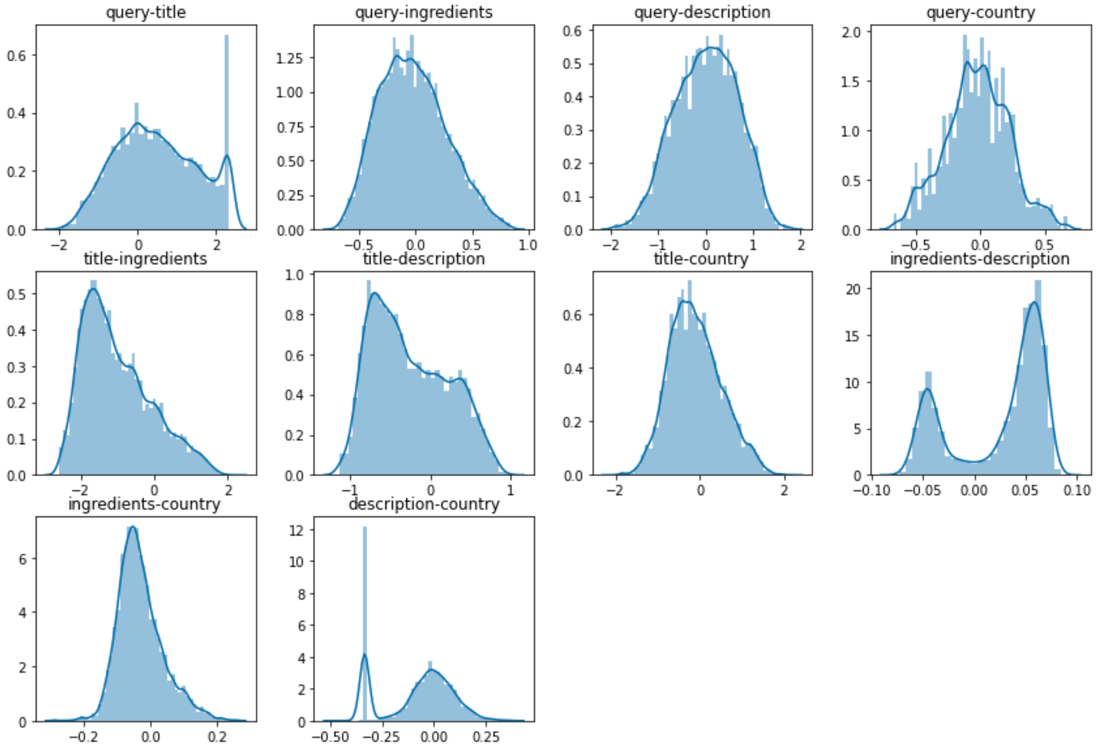


Figure 5.3: Distribution of outputs of the interaction layer

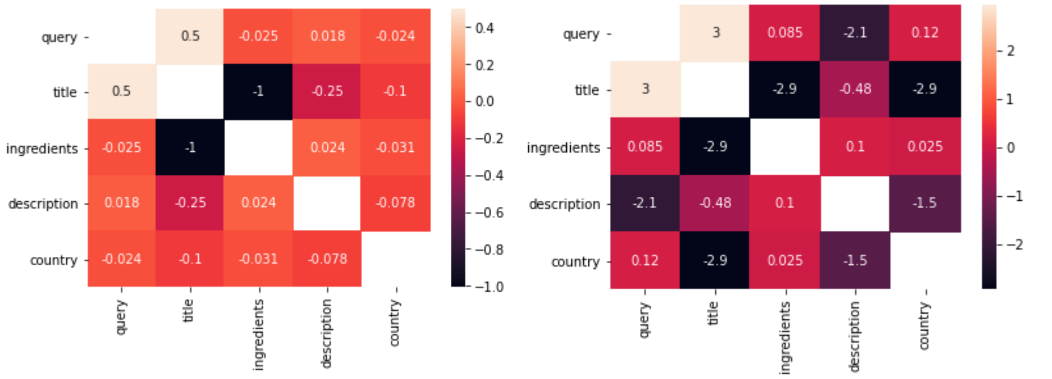


Figure 5.4: Heatmap of the strength of feature interactions

Chapter 6

Conclusion

WIP

6.1 Summary of outcomes

WIP

6.2 Summary of contributions

WIP

6.3 Areas for improvement and future work

WIP

- Better term matching
- Automatic feature interaction learning
- Personalization
- Deal with bias and noise
- Efficiency

- Index construction

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] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

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

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

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

[14] 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.

[15] 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.

[16] 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.

[17] Marco Ferrante, Nicola Ferro, and Silvia Pontarollo. Are ir evaluation measures on an interval scale? In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 67–74, 2017.

[18] 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.

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

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

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

[22] 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.

[23] Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364, 2017.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

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

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

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

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.

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

[51] 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.

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

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

[54] 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.

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

[56] 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.

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

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

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

[64] 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.