Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Jianghong Zhou Date

Improving Interactive Search with User Feedback

By

Jianghong Zhou
Doctor of Philosophy

Computer Science and Informatics

Eugene Agichtein, Ph.D.
Advisor

Joyce C. Ho, Ph.D.
Committee Member

Jinho Choi, Ph.D.
Committee Member

Surya Kallumadi, Ph.D.
Committee Member

Accepted:

Kimberly Jacob Arriola, Ph.D.
Dean of the James T. Laney School of Graduate Studies

Date

Improving Interactive Search with User Feedback

By

Jianghong Zhou
B.A., Sun Yat-sen University, China, 2017

Advisor: Eugene Agichtein, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2022

Abstract

Improving Interactive Search with User Feedback
By Jianghong Zhou

Capturing search users' feedback can improve the interactive search. In search tasks, users typically generate feedback while browsing the search results. That feedback may include clicking the items, reading important text content, query reformulation, and other interactions. This feedback can reveal users' latent intents and additional information needs, providing essential information to improve users' search experience.

Unlike traditional search, the interactive search is enriched by more interactions and comprises three significant steps: 1) Users browse the initial retrieval contents and generate feedback. 2) The feedback is received and analyzed by the search system. 3) The search system presents new search results based on users' feedback. However, the complexity of human interactions challenges these three crucial steps when building an efficient interactive search engine.

The first challenge is obtaining informative and valuable feedback from the users. This thesis introduces a new approach that can diversify the initial search results, allow users to explore multiple aspects of their original queries, and generate instructive feedback. The approach is the first to use Simpson's Diversity Index and Binary quadratic optimization in search diversification problems. Compared to the previous research, this method is more efficient.

Another key challenge is reducing the biased noise in the received feedback. In this thesis, we propose a novel de-biased method to decrease the feedback's high bias caused by users' observations. The approach uses a new observation mechanism to simulate the users' observation process and train a neural network model to detect the observation bias. This new model outperforms the previous click models in both click simulation and document ranking.

The last challenge is effectively extracting different interaction information and using them to improve the search. In this thesis, we focus on document-level and sentence-level interactions. We propose two different approaches with reinforcement learning frameworks. These methodologies introduce new techniques to reformulate the query and rank the items. Both methods significantly improve the search performance in the interactive search process.

Together these techniques provide imperative solutions to the challenges in the three critical steps of the interactive search systems and enable the users to obtain a better search experience.

Improving Interactive Search with User Feedback

By

Jianghong Zhou
B.A., Sun Yat-sen University, China, 2017

Advisor: Eugene Agichtein, Ph.D.

A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Informatics
2022

Acknowledgments

First and foremost, I am grateful to my esteemed advisor, Professor Eugene Agichtein, for his advice, support, and patience during my Ph.D. study. His immense knowledge and great experience have encouraged me throughout my academic research and daily life.

I want to thank all my teachers at Emory University for offering advanced courses necessary for my research and career. In particular, I would like to thank Dr. Joyce Ho, Dr. Jinho Choi and Dr. Kallumadi for being my committee members and offering invaluable suggestions.

My gratitude extends to all the members of the Emory IRLab. Their kind help and support have made my study and life a wonderful time, especially for my close friends Lin Chen and Ali Ahmadvand.

I have also greatly benefited from a summer internship at the Home Depot and being a visiting scholar at Johns Hopkins University Human Language Technology of Excellence. The supervision from Dr. Surya Kallumadi and Dr. Dawn Lawrie reinforce me to do more high-quality work and research.

Finally, I would like to express my gratitude to my parents, siblings, and friends. Without their tremendous understanding and encouragement over the past few years, it would be impossible for me to complete my study.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With the rapid development of the Web, the number of websites will be well over 1 billion in 2022 [54]. Therefore, it is almost impossible for ordinary users to locate their required information by simply browsing the Web. Consequently, search engines, efficient information retrieval systems, have become one of the most critical infrastructures in the information age [82]. Traditionally, search systems rank the documents based on the relationship between the items and the query, which can be relevance, preference, or significance. However, those search results are static and ignore users' feedback, which may lead to a poor user search experience. Due to the vast number of user interactions and the emergence of artificial intelligence, a search system that can serve users with more satisfactory results based on the users' interaction patterns is feasible and desired. In this thesis, I discuss this kind of novel search system, which is interactive search.

1.1 Interactive Search

Interactive search methods are the search systems that address the problem of finding the correct information based on an interactive dialog with the search system [130].

In the traditional search system, users submit their search queries to the search

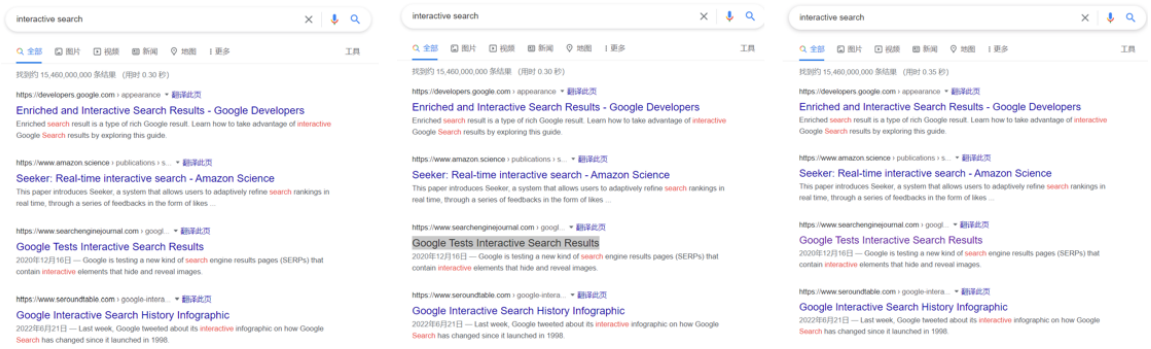

Figure 1.1: A traditional search system with static search results

systems, and the systems find their required information and present the search results to the users. This process is generally one direction. The search system does not consider the afterward feedback from the users and adjusts the search results. Those search systems are largely static: When the users' profiles are standard, their search results are identical no matter how the users interact with the search systems. Figure 1.1 shows a normal search system. A click of a less relevant result does not change the ranking result of the following search. This shows that traditional search systems could be more user-oriented. The optimization goal is primarily the relevance between the query and the documents, which may result in lower user satisfaction.

Besides, user interactions are common in many search tasks and rich in useful auxiliary information. For instance, in an e-commerce search, users may click on their interested items, browse reviews, or read product descriptions. That feedback is essential to discover users' preferences for the products. Additionally, users are likely only to give simple queries containing their primary intents without necessary details. Users' feedback can reflect that latent information and provide more accurate search results in limited presenting space. In summary, interactions provide extra information to improve the search results for the afterward or similar users' search. Therefore, capturing users' feedback to improve the search is significant.

Unlike traditional search systems, interactive search is rich in more interactions

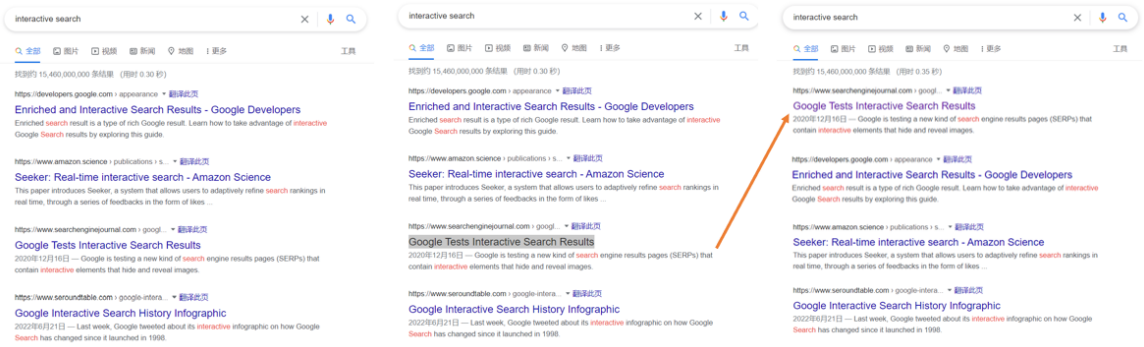

Figure 1.2: An interactive search system with dynamic search results

and has three major steps:

- The users submit an initial request to the search engine. The search engine retrieves relevant search results and presents the results to the users.
- The users browse the search results and passively evaluate the search results, which means the users do not necessarily process every document or all the information in the document. They generate feedback by clicking the items, copying the sentences, and other interactive behaviors.
- The search engine receives the feedback and adjusts the search results for the next or similar search session. Therefore, the search results of the interactive search are dynamic. Figure 1.2 is an example of an interactive search system.

With these three steps, the search systems can obtain additional information from the users and further satisfy their information needs. However, this process contains some critical research questions.

First, how do we present the search results so that the users are more likely to generate more informative feedback in the first step? The ways to present the search results are various, but we usually can only present a limited number of items to the users. If we only consider relevance, it is highly possible that the search systems only present similar results because the most relevant results typically share similar

essential information. For example, when the users search the term 'Apple', if we only present the most relevant results, it is highly likely that the search systems only present results related to Apple company, such as the products of Apple or the locations of their stores and introduction of this company. Those results only show one aspect of the search term and generate a filter bubble that limits the users from exploring other possibilities [101], so the users cannot generate any informative feedback. Therefore, it is essential to present diverse search results that provide different perspectives of the search query to explore users' information needs.

Second, how do we extract useful information from the feedback when the users browse the search results and passively evaluate the search results, which means the users do not necessarily process every document or all the information in the document. Due to this passive way, the most feedback we obtain in the search system is implicit [68]. The implicit dataset is not intentionally annotated by the users, which means the users do not observe some of the results, so the actual attitude to the searching items is uncertain. For example, the click logs are essential implicit feedback from the users. If the items are clicked, these search results are positive. However, if the search results are not clicked, they can be negative or just not observed. This kind of bias can lead to the mistaken label of disruptive noise. To effectively utilize the users' feedback, de-biasing the feedback is necessary. Therefore, reducing bias is the key to extracting useful information from the users' feedback.

Third, how can we integrate the feedback with the search system so that we can satisfy the users? Unlike traditional search systems, interactive search has more complicated information transmissions. The model patterning the interactive process should allow multi-directional information transmissions. A possible solution is accommodating the search process to a relatively dynamic system, such as reinforcement learning. Reinforcement learning can model a rather complex learning process and effectively reflect the interactions. Besides, different forms of feedback also bring

some challenges to the systems. The feedback of the users can be document-level, like clicking items. It can also be sentence-level, like selecting sentences. Different feedback may require different designs of reward functions, value functions, or policies.

Those challenges are essential to building an interactive search system. In this thesis, I contribute to building advanced interactive search systems by proposing novel and state-of-the-art solutions to those challenges, which I delineate in Section 1.6.

In summary, the interactive search can serve the users with dynamic results by collaborating users' engagements, resulting in three additional steps and extra challenges. All those challenges require practical and novel solutions.

1.2 Search Diversification

Diversifying search results is vital in many search systems due to the ambiguity of search queries and the complexity of users' intents. In multi-aspect search, a diverse set of search results are more likely to satisfy the users' needs and improve the online shopping experience. Therefore, many approaches for diversifying results in information retrieval and recommender systems have been proposed [154], among which the algorithms based on DPPs (Determinantal Point Processes) are considered to be state of the art methods. DPPs aim to balance diversity and relevance and are based on the geometric properties of vectors associated with a set of items [79]. DPPs define the diversity of a set by the volume of a parallelepiped spanned by the vectors of this set. The volume is determined by the vectors' length and their cosine similarity, which can represent the relevance and similarity, respectively. Specifically, DPPs compute the determinant of the product of the selected items' matrix and its transpose to identify a diverse subset of results.

Diversity is characterised by two aspects **1) richness** and **2) evenness** [121]. Richness quantifies the number of different class elements in a set, while evenness

considers the uniformity of the distribution of these classes. However, DPPs only consider the richness aspect [143]. It narrows the definition of diversity, which may degrade actual diverse search results. Besides, evenness can affect richness; especially if the item has multiple aspects [124].

In addition to not considering "evenness", DPPs also suffer from high computation complexity, which is NP-hard. In its greedy formulation, the computation complexity is $O(N^4)$, in which N is the scale of search space. Therefore, many variations are developed to improve its computational efficiency [27][28][78].

Due to the difficulties mentioned above, we consider another model to measure diversity, which is Simpson's diversity index [121]. It measures the probability that two items are chosen at random and independently from a set that is the same. Both Shannon entropy and Simpson's diversity index measure richness and evenness. Notwithstanding, Simpson's diversity index is a pairwise approach. Through some transformations and improvements, we develop it into a binary quadratic program with predicable approximation bounds.

We evaluate our approach on a standard benchmark dataset from Kaggle shoes' price competition (`https://www.kaggle.com/datafiniti/womens-shoes-prices`, `https://www.kaggle.com/datafiniti/mens-shoe-prices`). The experimental results show that the proposed method outperforms state-of-the-art DPP algorithms and extensions. In summary, our contributions are threefold:

1. Introducing Simpson's diversity index (SDI) to search result diversification domain.

2. We operationalize SDI computation as a quadratic program, which can be solved efficiently.

3. We show that the proposed method outperforms state-of-the-art methods on a standard e-commerce search benchmark.

Next, we motivate our method and place it in the context of related prior work.

1.3 User Feedback Modeling

To improve web search, it is essential to understand how the users interact with web search engines. Clicking on search results is one of the most critical interaction data, which forms the basis of many essential user behaviors, like reformulating or switching queries, clicking on different items, and browsing the search results. One powerful way to utilize these click logs is to construct a click model to measure and predict clicks on existing or future results[13]. A click model can predict future clicks of other users, help train a learning-to-rank (LTR) model, and allow us to automatically evaluate search result quality. However, modeling users' clicks is a challenging task because the click logs are observational data, collected in-situ with a live search engine, and exhibits multiple biases[69]. Previous research on click modeling and prediction did not directly address this issue or address the biases using heuristics, resulting in poor model performance on live (unseen) query traffic[69].

In summary, previous research focused on two directions. The first direction is the Probabilistic Graphical Model Framework Based Methods (PGM) [75]. Those methods model the search process as a sequence of events, predicting the click based on some probability models and assumptions. They are flexible and interpreted. The second direction considers a different representation of the events [13]. The search process is represented by some vectors. This form allows the users to consider a variety of features easily and to feed them to some relatively stronger learning models, such as neural networks [21]. However, the drawbacks of these two directions are obvious. The PGM methods are limited by a weak learning model and fewer features. The second direction cannot consider the bias issue in an interpreted way.

To overcome the aforementioned issues, we propose our new model, DRLC, for training unbiased (or less biased) click models, which is introduced in Chapter 4. DRLC is also a PGM-based method. Therefore, DRLC can be organized flexibly for different ranking scenarios and generate an interpretive model to reduce various

biases. However, in comparison to the previous PGM method, DRLC is constructed by a more dynamic system, which is reinforcement learning [127, 154, 156]. This allows DRLC to use stronger learning models (neural networks).

Another contribution of this work is that we propose a posteriori method to learn the observation bias. Previous approaches typically apply prior probability models to estimate the observation probability or reduce the bias [41, 3]. Those methods highly rely on the selected prior models, which is hard to be general. In this chapter, we concentrate on posterior knowledge. When users are browsing the SERP (search engine result page), the latter part is naturally unobserved. Therefore, if we use an observation window to augment the dataset, we can have some data with more bias and some data with less bias. This separation helps capture the bias. DRLC uses this method to reduce bias.

1.4 Dynamic Ranking

In complex search tasks, such as prior art patent searches, users require high relevance of the documents returned and the coverage of multiple perspectives on the topics. Therefore, users may continue interacting with the search engine. Simultaneously, the search engine should be able to mine the latent intents of the users from the interactions [146]. This is the basic scenario of dynamic search.

Figure 1 illustrates the dynamic search process. A dynamic search includes a search session containing many search iterations, whose results are a list of documents. Each search iteration includes many search units. The returned result of each search unit is a document. During the whole search session, users provide one query to the search engine and provide feedback after each iteration [73].

We formulate our approach based on the TREC Dynamic Domain setting [146]. In this setting, when a search session begins, a user searches for a query with several

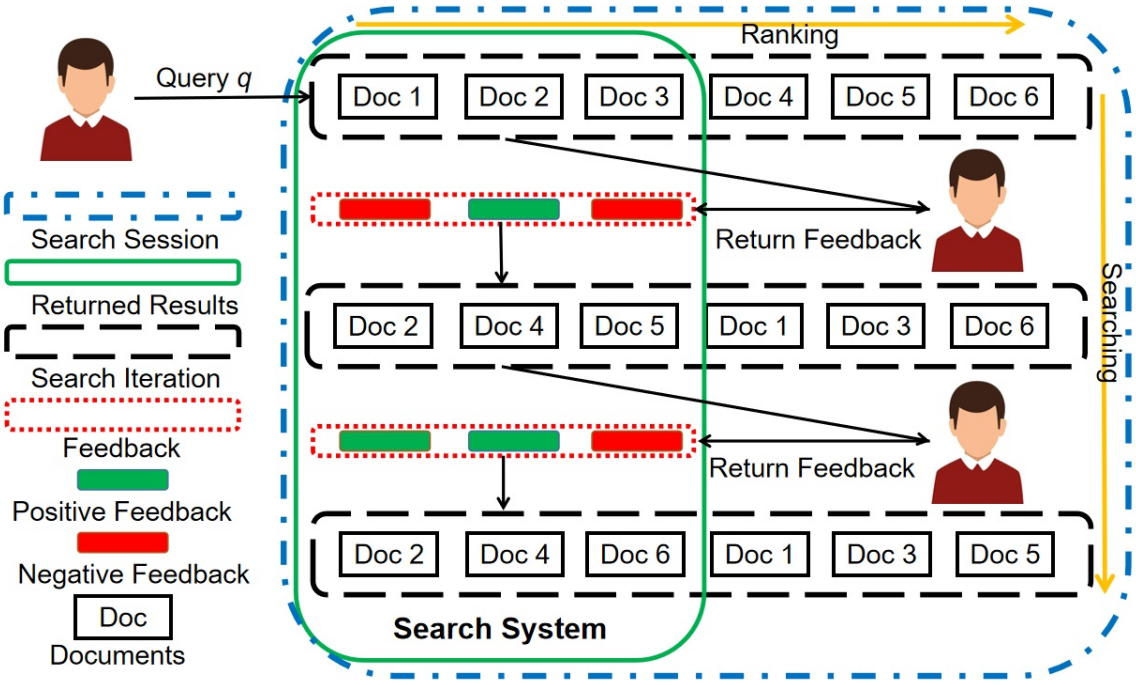

Figure 1.3: Flow of Dynamic Search with user feedback. The returned results are dynamically re-ranked based on the user's feedback.

The TREC domain Search Process.

subtopics which are not exposed to the search engine. This is the first search iteration. After that, the search engine analyzes the query and the documents' passages to rank the documents. Then, the search results are returned to the user, which is simulated in the TREC dataset. This simulator sends feedback to the search engine, indicating the relevance scores of each subtopic with the query. However, the feedback does not contain any information about the content or the number of subtopics. Based on the feedback, the search engine computes a score to evaluate this search iteration's quality, such as α-DCG [33]. After receiving the feedback, if the search session is not suspended, the search engine returns the new results based on the feedback. When reaching the stop conditions, the search session stops.

Based on this setting, extensive prior research studied the dynamic ranking mechanisms. In the 2016 TREC Dynamic Domain track, [7] introduced four models. The

highest performing model was **rmit-lm**, built by the language modeling approach from Apache Solr [48]. Another noteworthy model is **rmit-lm-oracle-1000**, which operates by retrieving the first 1000 documents, removing the irrelevant documents from the list using the ground truth, and returning the top 5 documents. This model was state of the art in the 2016 TREC Dynamic Domain track. However, this model uses ground truth labels to filter out documents that are usually unavailable in realistic tasks. Based on 2016 TREC Dynamic Domain, [145] applies the contextual bandit approach to dynamic search. However, the method does not outperform state of the art method in the 2016 TREC Dynamic Domain and lacks essential document ranking metrics in the experiments. In the 2017 TREC Dynamic Domain track, more models were proposed to improve the overall quality of the search session. Among those methods, **ictnet-params2-ns** is state of the art. Although those models present encouraging progress in dynamic search, they suffer some common flaws. Most of the aforementioned methods largely depend on some language models and traditional supervised Learning to Rank (LTR) models. The reported models were trained by pointwise, pairwise, or listwise methods. However, these approaches fail to capture all the ranked documents' information to improve the overall quality of the search session.

To address those problems, we propose a new method – RLIrank: Learning to Rank with Reinforcement Learning for Dynamic Search. RLIrank applies a reinforcement learning paradigm to conduct each search session of the dynamic search [127]. For each search iteration, we apply a step-wise training method to tune a stacked recurrent neural network, which explores and returns the search results [126]. We also apply an embedding-oriented Rocchio algorithm to digest the feedback [67]. The experiments on TREC 2016 Dynamic Domain and TREC 2017 Dynamic Domain show the RLIrank is a compelling dynamic search model, which outperforms the previous methods by up to 6.2%.

In summary, our contributions include the following:

- An effective implementation of the deep reinforcement learning paradigm adapted to the dynamic search problem, which captures the relationship between the user feedback and the document content.
- An effective deep value network model architecture, which incorporates a stacked LSTM and the associated new step-wise method to train the RLIRank model.
- Adaption of the classical Rocchio feedback algorithm to the deep RL framework, which extends the classic Rocchio algorithm to use dense word embeddings for relevance feedback in the neural ranking setting.
- Evaluation of the RLIRank method on the 2016&2017 TREC Dynamic Domain datasets, showing that RLIrank outperforms all previously published methods in these two tracks after as few as five search iterations.

1.5 Interactive Search with Sentence-level Feedback

Search with interactive settings can improve the performance of an information retrieval system [111]. Users' interaction not only supplements the information of the query but also conveys users' latent intents. Users' feedback involves item-level feedback (e.g., item clicking, purchasing, and adding to cart) and sentence-level feedback (e.g., sentence clicking, copying, or reading). Sentence-level feedback can provide more details than item-level feedback. For example, in Figure 1.4, User 1 searched 'How did people survive from the 2011 Fukushima earthquake' and selected (e.g., reading, clicking, or copying) a sentence from the 3rd document as a piece of essential information. This interaction revealed that User 1 was looking for stories of

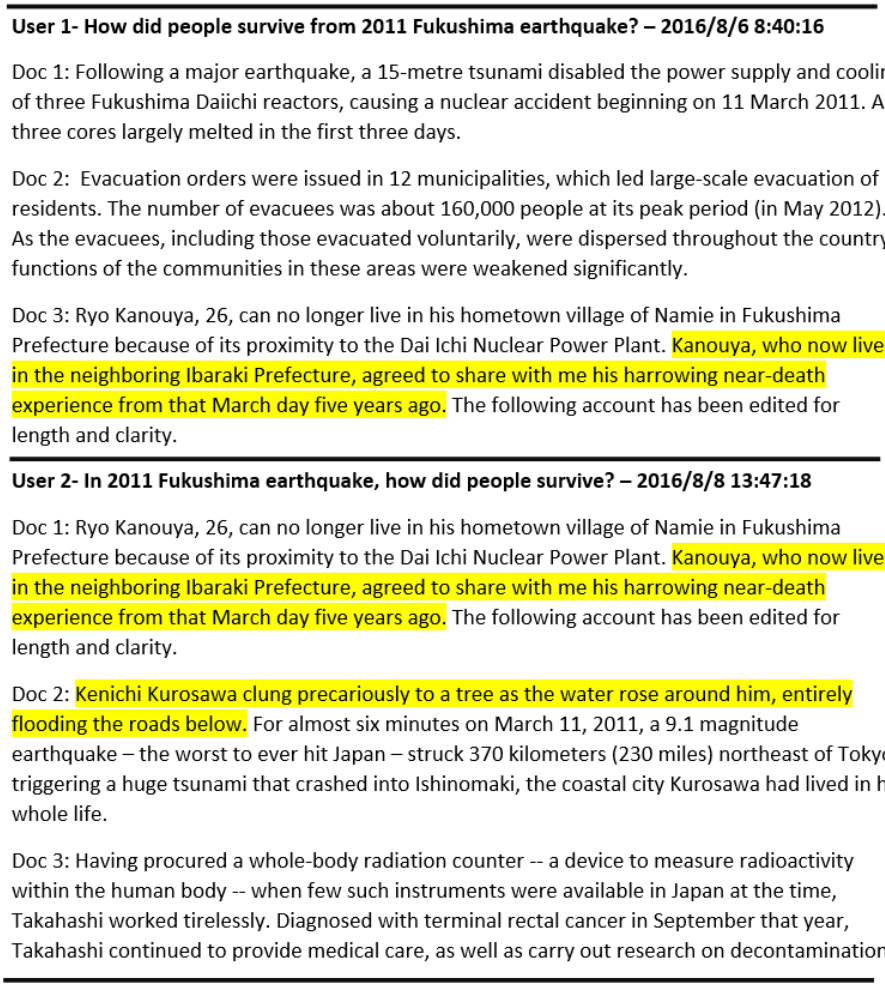

Figure 1.4: A use case of interactive search

survivors, helping User 2, who searched a similar query and obtained new results containing those stories based on User 1's feedback. When considering many people's interactions, the search results can approach the human's expected results. This scenario is common in question-answering systems (QA), e-commerce product reviews, and other complex search tasks [87, 156].

To build an interactive search environment, reinforcement learning (RL) approaches are the most common choices in the previous works [153]. However, these studies suffer from two imperative challenges relating to (1) the amount of data needed to train the RL model and (2) the computational power needed to search the ample ranking space. We propose DQrank, a deep Q learning (DQ) based ranking system, to address

these limitations.

First, obtaining data to train an interactive search model is strenuous due to the dynamic search environment. Directly depending on online users' interaction is costly and time-consuming, as users need to interact multiple times with the system to yield even semi-reasonable ranking results. On the other hand, solely relying on offline, static interaction records with only a limited set of all possible ranking results restricts the exploration of interactive search [112]. Expanding the static interaction logs and combining these two learning ways are necessary. Previous research used strong assumptions to connect new ranking results to existing ones. [140] discounted or boosted the relevance between the query and document based on the position distance between the document and the clicked one. However, this approach can only relieve the effect of position bias but fail to reduce other bias, such as observation bias [157]. [5] estimated the propensity to approximate the real relevance between the document and query, but still required encoding of the heuristic knowledge and can yield other biases. Furthermore, these methods focus on expanding the item-level connection, which cannot work on sentence-level feedback directly. In this thesis, we reduce the need for annotated data by adopting a BERT model [89] pre-trained using tasks with explicit feedback to simulate user interactions. This user simulation function can then re-rank the items to obtain more satisfactory results. We also propose a novel state retrieval approach that combines both offline and online learning. By using similar queries as the starting point for RL, DQrank can obtain a better initial ranking performance.

The second challenge, different from most RL tasks, lies in the notoriously gigantic action space of the ranking, so optimal exploration is infeasible. Previous research proposes different methods to approximate the optimal action that maximizes the target score. For example, RLIrank greedily ranks the documents with a neural-based agent [154]. SlateQ decomposes the ranking to build their item choice model with

mild assumptions [61]. These methods can only obtain sub-optimal actions based on their choice mechanisms. Besides, they reformulate the ranking problems to point-wise problems without considering the effect of the surrounding items, which loses essential auxiliary information for the ranking evaluation and ignores widespread bias in real-life search tasks. To address the problem, we introduce a sliding window ranking method and rearrangement learning to explore the optimal actions. The sliding window prioritizes the top-ranking documents by focusing on a small range and pushing the significant items to the top position. Additionally, rearrangement learning can adaptively learn the preferred order and continually improve the subsequent rankings.

In summary, DQrank's contributions are three-fold:

- DQrank contributes new methods, including self-supervised learning, BERT-based users' simulation, and state retrieval, to train the interactive search models, which expands the static search logs for the dynamic search environment and integrates offline and online learning.
- DQrank introduces novel sliding window ranking and rearrangement learning to efficiently explore the ranking results that can maximize the long-term value in DQ without losing the list-wise information of the ranking systems.
- Additionally, we demonstrate our methods in open standard datasets and study the insights in a detailed ablation study.

1.6 Research questions and Contributions

Based on the above introduction, three fundamental research challenges are identified as follows:

- RQ1: How can we effectively diversify the search results for more informative feedback?

- RQ2: How to de-bias the feedback information to better extract useful information?

- RQ3: How can we model the interactive search process so that we can use feedback to improve the search?

The first two steps are the necessary infrastructure for interactive search. The RQ3 provides methodologies to build interactive search systems.

In this thesis, I present new algorithms and techniques to address these challenges in four chapters. In Chapter 3, we will introduce an SDI-based search diversification method, which can efficiently improve the diversity of the initial search. In Chapter 4, I propose a novel user simulation to reduce the bias in user feedback. In Chapter 5, I introduce my solution to use the document level to build the interactive search system. This reinforcement learning-based approach outperforms the previous research. In Chapter 6, a deep Q learning-based interactive search system is presented to effectively process sentence-level feedback, which achieves SOTA performance.

1.6.1 An Efficient Approach to Diversify the Initial Search for More Informative Feedback

As we mentioned, diversifying search results helps obtain informative feedback. The first step is to measure the diversity of the ranking results to achieve this. An interpreted metric of diversity can help us observe the diversity of the ranking results and also path a way to optimize the ranking results. We decompose this challenge into two sub-problems:

- 1. Modeling the diversity of the ranking results.

- 2. Optimizing this model to improve the diversity of the ranking.

In previous research, Determinantal Point Processes (DPPs) are proposed to measure the diversity and diversify results. DPPs use the volume of a parallelepiped spanned by the vectors of the items' attributes to measure the diversity. The volume is determined by the vectors' length (relevance) and cosine similarity (similarity). By selecting a subset to maximize this volume, we can maximize both the diversity and relevance of the ranking results. Although DPPs are effective in many applications, they are suffered from high time complexity, uncertain optimum of the approximate methods, and instability. To overcome those difficulties, we consider other simpler models with concrete mathematical solutions. A highly competitive is Simpson's diversity index (SDI). Simpson's Diversity Index (SDI) was initially introduced in biology to measure the evenness and richness of a habitat [121] and aims to estimate the probability that two randomly selected individuals from a sample will be the same. SDI can be reorganized as a binary quadratic optimization problem with concrete solutions and promises sub-optimum. To verify the proposed method, we will use a Kaggle shoes challenge dataset and some important metrics, such as coverage, α-NDCG, and time complexity, to demonstrate the effectiveness of the proposed method. We can compare the proposed method with the previous research with those experiments, validating its effectiveness. In this research, our contributions include:

- We introduce an effective diversification model SDI to measure diversity in the search problem from biology.
- We use the binary quadratic optimization method to optimize this model efficiently.

  The details of this contribution are described in Chapter 3.

1.6.2 A New De-biased Method for modeling search Feedback

In terms of the second research challenge, we first analyze how the search bias generates and then build the models to reduce bias.

Initially, the users send the query to the search engine. Then the search engine presents the search results. After that, the users read some of the results and send back feedback. In the third part, the information flow is, in fact, incomplete and may lead to unpredictable data issues because the users only receive a small part of the search results, and the pattern of how the users select the information is uncertain. The users passively receive the search results. Therefore, when users select valuable information as feedback, they do not select the information from all the search results but from their observed results. Therefore, the feedback is unavoidably biased. To extract the helpful information from the feedback better, we need to de-biased the feedback. If we ignore the unobserved positive feedback or consider it negative, it will lead to incorrect annotations and massive noise.

Therefore, it is imperative to develop a method to reduce those biases. A straightforward method is to detect those bias. When the bias is detected, we can choose to remove them or reduce its effect. However, since the bias is passive data, it is difficult to directly evaluate the accuracy of the detection or measure the improvement of the de-biased feedback.

In summary, we have two major questions that need to be answered:

- Developing a method to detect the bias.
- Validating the model can be used to reduce the bias and improve the feedback quality.

To de-biased the feedback, two significant courses can be considered. We can use

statistical methods to balance the bias. Second, we can model the users' observation patterns to discover the bias from the dataset.

The first kind of method typically still considers specific patterns of the users' observation—for example, the distance between the positive items and negative ones. For the second course, validating that the model can effectively reduce the bias is hard theoretically. Considering the drawbacks of these two courses, we propose a mixed approach.

We use reinforcement learning and convolution neural networks to simulate the users' behaviors and learn the bias patterns. The proposed method can generate both de-biased feedback and biased feedback. Therefore, we can compare the ranking performance by training the ranking models with additional feedback in our experiment. Simultaneously, we also calculate the recall of the unbiased results to demonstrate that the proposed model can effectively simulate the users.

We further design experiments to demonstrate that the proposed method can improve the quality of the feedback by improving the ranking results. In terms of experiments, we use two public datasets: ORCAS dataset [37] and Yandex click dataset [117] and a real interactive dataset from a large e-commerce website (`https://www.homedepot.com/`) to validate the effectiveness of the model.

The experiment mentioned above can answer the research question in two aspects. First, the feedback is biased, and de-biasing the feedback can improve the performance of ranking models. Second, the patterns of the bias can be learned and simulated. De-biasing the feedback can avoid possible mistaken annotations and massive noise, enhancing the afterward search performance. Therefore, this research has three significant contributions:

- We can theoretically and practically demonstrate that the passive bias in feedback data can degrade the performance of the ranking models. It is vital to de-bias the feedback in interactive search systems.

- We propose a possible solution to de-bias the feedback data. The proposed model simulates the users' bias patterns, which can de-bias the feedback data and be used to select learning to-rank models.

- In the future, we may extend the model to analyze users' bias patterns and extract the intent information further.

We demonstrate this contribution in Chapter 4.

1.6.3 Novel effective reinforcement learning approaches for interactive search with different level feedback

After solving the last two key challenges, we can effectively use the feedback information to improve the subsequent search performance. We propose new reinforcement learning methods to integrate feedback information into the interactive search systems in this part.

First of all, interactive search systems are dynamic. The search system needs to adjust the search results based on the reaction of the users when they receive the previous search results. Figure 1.5 is the diagram of interactive search.

This scenario can be modeled by reinforcement learning. In this thesis, we model interactive search systems with reinforcement learning. However, different interactions have different commuting mechanisms, resulting in different environments, policies, and reward settings. The methods of reinforcement learning methods also can be various. Therefore, the feedback forms must be carefully considered when building reinforcement learning-based interactive search systems.

Generally, we need to consider two forms of interaction. First, the feedback is document-based. In this situation, the users' feedback only contains information about their attitudes to some documents. Some of the documents are positive, some of them are negative, and the rest of them are neutral. Under this circumstance, the

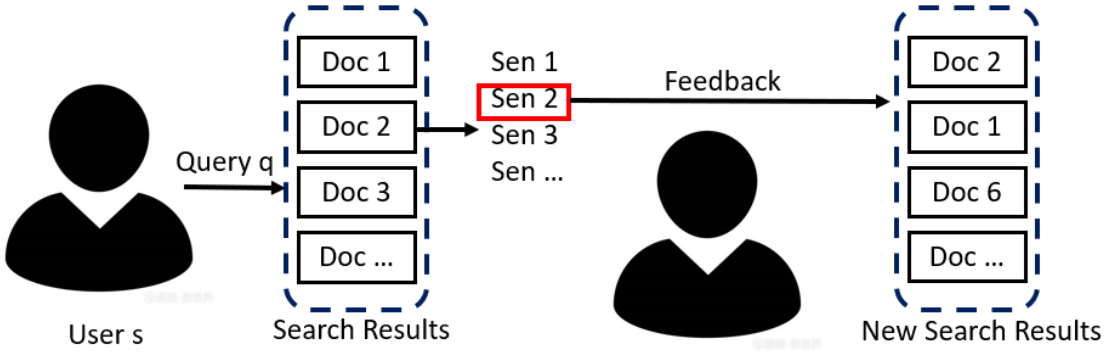

Figure 1.5: Interactive Search Diagram

feedback information is straightforward: some documents should be ranked ahead, and some should be moved back. Therefore, document-level feedback is position information.

The second form of feedback is sentence-based or text-based. The users select some of the sentences in some documents as feedback. In contrast to document-level feedback, sentence-level information is more specific. They may provide clues about users' latent intent or give more details to expand the original query. In summary, sentence-level feedback is supplementary material for the query.

Considering these two basic forms of feedback, we have two subtopics to discuss in this Section:

- 1. Building a reinforcement learning-based interactive search system whose feedback is document-based.
- 2. Building a reinforcement learning-based interactive search system whose feedback is sentence-based.

We consider deep reinforcement learning as the framework to model the interaction process in both situations. However, their embedding models, the value network, and the updating methods are different.

A Novel Document-based Interactive Search System

Regarding document-based interactions, the critical feedback information is that the documents are misplaced. Some documents should be placed ahead, and some should be placed back. Therefore, adjusting the ranking scores is the key task in the document-based interactive search system. We utilize a policy gradient reinforcement learning framework to optimize the estimated ranking scores for the documents, which is the first attempt to build the interactive search system with the policy gradient methods [102]. In this framework, we can encourage the search agent to increase the probability of high-score ranking results and discourage low-ranking results.

Additionally, we adapt the Rocchio algorithm to cooperate the feedback information with the query in the search session and propose a step-wise learning framework to optimize the list-wise search metrics [67][119]. All those efforts enable us to efficiently extract the position information from the document-level feedback.

Moreover, we use four public ranking datasets to validate the proposed search systems. Two standard-ranking datasets, MQ2007 and MQ2008 [105]. These two datasets are used to demonstrate the performance of the initial ranking. The other two are dynamic search datasets: TREC Dynamic Domain 2017 and TREC Dynamic Domain 2018 [150]. We use these two datasets to show that the proposed system can improve search performance based on users' document-level feedback. The experiments on those datasets demonstrate the efficiency of the proposed search system.

We further delineate the system in Chapter 5.

A State-of-the-art Sentence-level Feedback Interactive Search System

In the sentence-based system, since the feedback is text content, it shares a similar information form as the query. Therefore, we can integrate the feedback information into the query and directly optimize the ranking orders of the documents.

Based on the above consideration, we build a deep Q learning-based interactive

search system to utilize the sentence-level feedback to improve the search performance [42]. In this system, we build a BERT-based Q function to evaluate the actions, which are different ranking results given the query and the sentence-level feedback.

However, we still have other challenges in this framework. One is the document selecting model. Because the number of document ranking results exceeds the practical scope, we design a new selecting method to estimate the ranking results with the highest Q value. Besides, we also introduce text content augmentation techniques to stabilize the training process. Together, all those innovations allow the proposed system to achieve state-of-the-art performance in sentence-level interaction search systems.

In the experimental part, we use different datasets to fine-tune the pre-trained BERT model and validate the systems with the TREC CAST dataset, the MS-MARCO dataset, and the BETTER dataset. Compared to the previous approaches, the proposed system is state-of-the-art.

We present details of the proposed system in Chapter 6.

In summary, this part demonstrates using reinforcement learning and different embedding methods to improve the search with different feedback forms. We can demonstrate that reinforcement learning is a practical framework to model the interactive search process. We provide two effective models for two different basic situations of interactive search. In the future, these two situations can be mixed and considered another kind of feedback information organized as a knowledge graph.

This thesis proposal presents three critical questions and their solutions for improving the search with feedback. First, we diversify the search results to help the users generate helpful feedback. Second, we de-bias the feedback data to extract useful information better. Third, we apply deep reinforcement learning and different embedding methods to utilize different feedback information.

Before introducing the new algorithms and techniques, I first review related work

about user feedback and search in Chapter 2.

Chapter 2

Background

This work of this thesis focuses on building efficient interactive search systems, and we have three major goals:

- to diversify the search results so that the users can generate useful feedback.
- to reduce the bias caused by users' behavior patterns in the feedback.
- to use the feedback information to improve the search performance.

Those goals rely on analyzing basic learning to rank models, search bias modeling, and interactive search systems. In this chapter, I review related work on these previous research:

- Interactive search (Section 2.1),
- Learning to rank (Section 2.2)
- Search diversification (Section 2.3),
- User feedback modeling (Section 2.4).

2.1 Interactive Search

With the immense amount of data on the web, it is significant to develop efficient search services to retrieve users' desired information. An efficient way to understand users' information needs is by analyzing users' interactions like clicking, observing, attention, query reformulation, category selecting, reviewing, and other behaviors. Those engagements can reflect users' satisfaction with the search results and convey users' latent intents. Undoubtedly that information helps improve the subsequent search. A better subsequent search can further boost the next search. This is the basic intuition of interactive search. We can summarize the interactive search as the following equations:

$$S_0 = f(q, 0) \tag{2.1}$$

$$S_{n+1} = f(q, u(S_n)) \tag{2.2}$$

where S is the search results, f is the search engines and u is the users' feedback and q is the initial search query.

We can observe that 2.4 is the traditional one-time search system. In the interactive search, we assume the search results are dynamic, and the users' feedback can improve the next search:

$$T(S_n) \leq T(S_{n+1}) \tag{2.3}$$

Where T is a function to evaluate the search results, which can be Normalized Discounted Cumulative Gain (NDCG), Mean reciprocal rank (MRR), click-through rate (CTR), revenue, and other metrics [8, 137].

In this case, we assume interactive search is a positive feedback mechanism [86]. We may consider a convergence like:

Figure 2.1: The framework of the interactive search. The figure is from [59].

$$T(S_{n+1}) - T(S_n) \leq \epsilon \tag{2.4}$$

Where ϵ is a threshold of the convergence, however, this can be only the short-term optimization. The evaluation of the search results can be dynamic at different times. For instance, 'Christmas Tree' can be a good search result during winter for the query 'tree'; in Autumn, 'Maple tree' is a better result.

Therefore, interactive can not only capture users' intents in a search session but also can automatically adjust the search results and follow the trend based on users' up-to-date feedback.

Additionally, the interactive search can be applied to many services, like keyword search engines, semantic search engines, question-answering systems, dialogue systems, and chatbots [4], because they are abundant with interaction signals. An example of interactive search is presented in 2.1. Users' clicking behavior helps the keyword search engine generate better search results. Interactive search is a universal framework that can be applied to most search systems.

In summary, the advantages of interactive search are three-fold:

1. Interactive search is a positive feedback system which promises improved subsequent search.

2. Interactive search can satisfy the users in short-term and long-term engagement.

3. Interactive search is a universal mechanism that can be applied to most search services.

2.2 Learning to Rank

Learning to rank (LTR) is the foundation of all search tasks, including interactive search. It is a technique to construct a ranking model that can sort items based on their relevance, preference, and importance [88]. It is crucial to many information retrieval tasks, such as search, recommendation systems, and chatbot [92] [40] [63].

Learning about LTR's basic concepts and evaluation methods is imperative to understand this thesis. In this section, I introduce the basic ranking methods, the evaluation approaches of LTR, and three basic styles of LTR: the point-wise, pair-wise, and listwise approaches.

In ranking tasks, different approaches may rank documents based on different criteria. When focusing on the relationship between the query and the documents, they are query-dependent models. The other methods rank the documents based on the importance of the items, which are categorized as query-independent models.

2.2.1 Conventional ranking models

Sorting the documents based on the frequency of the query terms in the documents is one of the simplest methods for learning to rank. However, the occurrences of the terms cannot fully measure the relevance. A better idea is building a representation of the documents and queries, which can reveal their more profound connection.

With this primary thought, some models based on vector space are proposed [9]. In those methods, the text content of both queries and documents is encoded as vectors. They can further use the inner product, Euclidean distance, and Manhattan distance to measure the relevance between the vectors of the document and query

[30].

One of the most common methods to represent the terms is TF-IDF weighting [6]. The TF of term t in a vector is the normalized number of its occurrences in the document, and the IDF can be defined as:

$$IDF(t) = \log \frac{N}{n(t)} \tag{2.5}$$

Where N is the number of documents. $n(t)$ is the number of the documents containing term t.

While TF-IDF can represent the terms better than their frequency, it assumes that all the terms are independent, which is not necessarily correct. Some approaches, like Singular Value Decomposition, are used to linearly transform the vector space, which can reduce the dependency of the terms.

Except for calculating similarity, other approaches are developed to measure the relevance, such as BM25 and language models. BM25, or Okapi BM25, is a probabilistic-based model which estimates the probability that the documents should be ranked ahead [110]. The score of BM25 is defined as:

$$BM25(d, q) = \sum_{i=1}^{M} \frac{IDF(t_u)TF(t_i, d)(k_1 + 1)}{TF(t_i, d) + k_1(1 - b + b\frac{l(d)}{a})} \tag{2.6}$$

Where q is the query, containing terms t_1, t_2, ..., t_M, d is the document. $TF(t, d)$ is the term frequency of t in document d. $l(d)$ is the number of words of document d. a is the average document length of all documents. k_1 and b are two parameters. $IDF(t)$ is Equation 2.5.

BM25 is a classical method to rank documents. In this thesis, we use BM25 to retrieve candidate documents and then use our method to re-rank the document. BM25 can rank documents efficiently but are weak in ranking long documents [131]. Except Equation 2.6, BM25 also has many other variations [72].

The language model is another branch of learning to rank based on statistical methods. In language models, documents are ranked based on the probability that the terms appear in the query giving the document (i.e., $P(q|d)$), where q is the query and d is the document. If we assume the terms in the query are independent, then we have the following:

$$P(q|d) = \prod_{i=1}^{M} P(t_i|d) \tag{2.7}$$

where query q consists of terms $t_1, ..., t_M$.

The maximum likelihood method is further applied to estimate the language models. Previous research of language models discovered that a background language model is required for smoothing the estimation [103], which further constructs the model as:

$$p(t_i|d) = (1 - \lambda)\frac{TF(t_i, d)}{len(d)} + \lambda p(t_i|C) \tag{2.8}$$

where the background language for term t_i is defined as $p(t_i|C)$ and $\lambda \in [0, 1]$ is the smoothing factor.

Those methods are essential foundations for search. They are usually swift and widely used in retrieving candidate documents for the next round of re-ranking, which will re-rank the candidates to consider additional criteria or constraints from an initial pool of candidates.

2.2.2 Point-wise, pair-wise and list-wise models

The learning-to-rank methods can be categorized into point-wise, pair-wise, and list-wise methods based on different hypotheses and loss functions.

The point-wise models

The point-wise method assumes that the documents in the rank list are independent. The point-wise models estimate the relevance degree of every single document with

the query and then rank them based on the scores. The input features are extracted from the query, a single document, and their cross relationship.

Therefore, the task of point-wise ranking methods is training a score function, which is straightforward and highly connected to the traditional machine learning approaches. Generally, the point-wise loss function examines the prediction of some ground truth label, such as click-through rate, review score, or clicking. Consequently, regression loss and classification loss can be used as loss functions. Those machine-learning approaches have developed mature solutions. For example, XGboost is a widely used point-wise method in the industry [29].

However, limited by its assumption that the documents are independent, the position information is invisible to the point-wise method. At the same time, it also ignores the bias effect of the ranking, which means the users have different relevance evaluations when the documents appear in different positions [69].

The pair-wise models

The pair-wise models compare the relevance between the two documents. By comparing every two documents in the candidate lists, those documents can be ordered.

Unlike point-wise approaches, the input of the pair-wise documents contains a pair of query-document features. The output is $\{1, -1\}$, demonstrating the preference of these two documents.

Therefore, the loss functions of these methods focus on the inconsistency between the estimated score and the ground truth label. For example, considering the click-through rate as the ground truth label, we may need to set a threshold ϵ to generate a label for every pair of documents. For example, let us assume the click-through rate of document i is C_i and the click-through rate of document j is C_j. The new label $D(i, j)$ will be:

$$D(i,j) = \begin{cases} 1, & C_1 - C_2 \leq \epsilon \\ -1, & C_1 - C_2 > \epsilon \end{cases}$$

Therefore, the pair-wise models can be simplified as a classification problem. The pair-wise approach helps prioritize the relevant documents by effectively distinguishing the irrelevant results and putting them in the latter positions.

Although pair-wise models consider the relative relation between every two documents, some drawbacks are still apparent. First, the pair-wise models cannot reflect the position information fully. It only reveals the relationship between every two items. Second, the relevance is not singular, which means the query and the documents can be relevant in different aspects. For example, we may have $D(i,j) = 1$ and $D(j,k) = 1$, but we cannot directly have $D(i,k) = 1$. We can have $D(i,k) = -1$. This contradiction is led by the inconsistency of the relevance and may need additional mechanisms to justify the correct order [56].

The list-wise models

The list-wise models evaluate the ranking performance of the ranking results. The input is the entire group of the ordered documents associated with the query.

The output of the list-wise models can be a list of scores, which demonstrates the relevancy of every document in the ranking list, or a score representing the relevance degree of the whole list with the query.

Unlike point-wise and pair-wise models, list-wise models consider the positions and the inter-dependency among all documents, which can capture more information in the search tasks. However, most search tasks do not evaluate the whole ranking list directly because this can cause the loss of essential annotations. For example, evaluating the ranking results by only considering whether the click happened but not capturing which document is clicked may make our annotations less specific. To

resolve this problem, we can use some metrics to evaluate the ranking and consider them as the loss function. For example, we can use Normalized Discounted Cumulative Gain (NDCG) to calculate the performance of the ranking results [132].

NDCG measures the quality of a set of search results and has the following form:

$$DCG@k = \sum_{i=1}^{k} \frac{D_i}{\ln i + 1} \tag{2.9}$$

$$NDCG@k = \frac{DCG@k}{iDCG@k} \tag{2.10}$$

Where k is the number of documents in the search results. $iDCG@k$ is the $DCG@k$ of the best ranking. D_i is the relevance score of the document i.

These three learning-to-rank models are the basic frameworks of most search tasks. We will introduce some work closely related to interactive search in Section 2.2.3.

2.2.3 Learning-to-Rank (LTR) methods

In previous research summarized in [88], most of the methods to solve Learning-to-Rank problems are point-wise, pair-wise, or listwise approaches. The point-wise approaches use a single document as its input in learning and define the loss functions by the relevance of each document. In contrast, pair-wise approaches take document pairs as instances, while listwise approaches consider the whole list of documents [84][82][81][15][144][16].

More recent research on learning-to-rank paid more attention to the extra information provided by multiple documents and thus is majorly pair-wise methods or listwise methods. For example, by formalizing the learning-to-rank problem as a problem of binary classification on document pairs and applying a Support Vector Machine (SVM) to solve this classification problem, Joachims et al. proposed a pair-wise method–RankSVM [81]. By using the classification model as a Neural Net-

work instead, Burges et al. proposed another pair-wise method, which is RankNet [15]. In terms of the listwise method, the most noticeable example algorithms include AdaRank based on NDCG and ListNet. AdaRank-NDCG is a method that repeatedly constructs weak rankers based on re-weighted training queries and finally linearly combines the weak rankers for predicting the rank, in which the weaker ranker is measured by NDCG [144]. ListNet is based on a probability distribution on permutations [16]. Specifically, for a document list, ListNet applies the Luce model to generate a permutation probability distribution based on the scores outputted by the ranking function. Then, it uses the same method to generate another distribution based on the ground truth labels. After that, two distributions' cross-entropy construct the loss function.

Although those approaches greatly model different essential features of LTR, those methods are still restricted to a specific data structure. In fact, during the search process, the length of the rank is increasing. None of the aforementioned methods models this important dynamic process. Inspired by this, we further extend the listwise method to the stepwise method.

Inspired by the Bellman function in Q learning and the structure of LSTM, we propose a stepwise method – sLSTM, which is a new and better LTR model [138][49][10]. sLSTM utilizes every ranking results from a search iteration. For example, when the sLSTM ranks the first document, the training process considers this is a document list with one document. Based on the ranked document, sLSTM ranks the second document, and the training process considers it a new document list with two documents. In the discussion part, we reveal that the document lists with the different numbers of documents enrich the training dataset. The shorter document lists are not just part of the longer document lists; they present some different information, lead the model to avoid useless aspects of ranking, and are helpful in avoiding over-fitting, which has a similar effect to data augmentation in digital image processing.

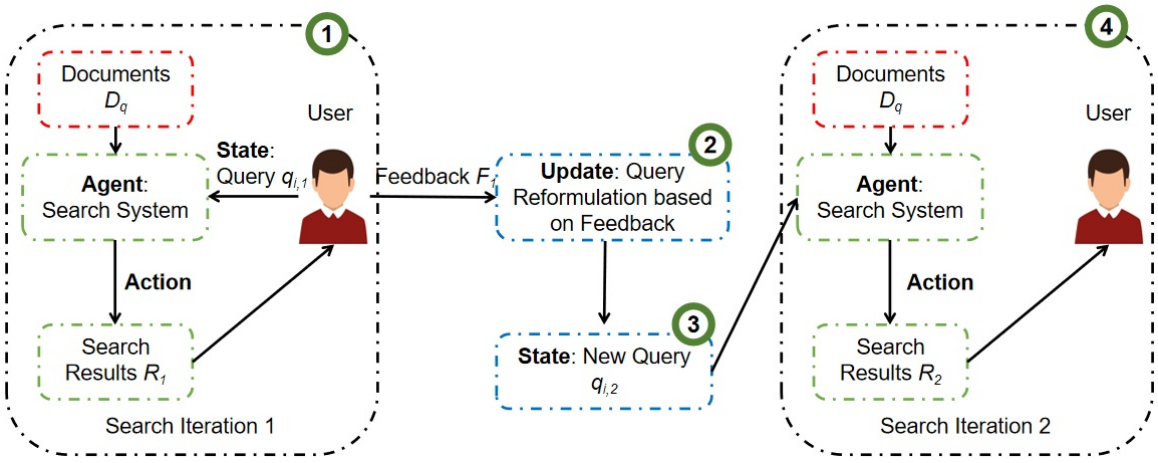

Figure 2.2: The framework of the dynamic search model.

2.2.4 Reinforcement Learning-based Methods

Reinforcement learning (RL) optimizes an agent policy to interact with the environment.

The application of RL in search needs to consider the search as an interactive process. One crucial consideration is relevance. Relevance is not an objective label. Users' behaviors during searching, like query formulation, document clicks, document examination, eye movement, mouse movement, etc., typically expose their natural preferences, which may differ for each user. Therefore, the information from users during the search can be valuable to guide the rest of the search process, which could lead to better overall performance. This kind of ranking can be considered partially interactive. Although this ranking considers the users' preference, it does not consider the effect of previously seen ranked documents and the overall performance of the whole ranking. Compared with partial interaction ranking, dynamic ranking considers both interaction and long-term gain.

The basic strategy of dynamic ranking is determining the importance of different information while helping the users explore the information space. Therefore, dynamic ranking contains four major characteristics [147]. First, dynamic ranking optimizes all interactions. Second, it considers long-term gain. Third, it can predict

future users' preferences. Four, it affects the beginning of the interaction.

Recently, many approaches to LTR have been proposed based on reinforcement learning. The early research on applying reinforcement learning in the ranking is [95], which uses Partially Observable Markov Decision Processes (POMDPs). In 2013, Guan et al. further developed it into a session search algorithm, which improves the performance of the MDP by attaching different importance to different time documents in the reward function [50]. Expanding on these two methods, Glowaka et al. designed a reinforcement system supporting the active engagement of users during search [47].

Most recently, and closest to our work, is a recently introduced method that uses Reinforcement Learning to Rank with Markov Decision Process (MDP) [139]. One of the central ideas of this method is treating the document list as a sample instead of evaluating documents one by one. This allows MDP to optimize performance evaluation metrics, like NDCG score directly. To generate a document rank, the authors apply the Markov Decision Process to rank the documents first and then further train the model by the whole ranked documents list. As a result, the MDP method was demonstrated to be the new state-of-the-art on the MQ2007 and MQ2008 benchmarks. In follow-up work, [44][85] successfully applied MDP to search diversification problems. Those results are essential for dynamic search problems. Dynamic search has a complex learning process, so it usually requires constructing an environment and modeling the interaction between the users and the search engine. Those processes can be modeled by different roles in reinforcement learning-based algorithms. Therefore, reinforcement learning is also a suitable framework for dynamic search.

Specifically for the dynamic search setting, most prior research focuses on the framework design, such as win-win search and Partially Observable Markov Decision Processes (POMDP) [90][148]. To compare different frameworks' effects, [129] explores different reinforcement learning-based frameworks in dynamic search, in-

cluding Markov Decision Process, Q learning, and Deep Q learning network. Those results are included in the 2017 TREC Dynamic Domain track. In the deep Q learning network method, LSTM is used as a data processor, and the Bellman function as the reward function. They also utilize a naive method to update the query by adding or removing some words based on feedback scores on each result. Although the framework is important to improve performance, the reward function and the way to update the query are also significant. In Chapter 5, we design a deep value network to estimate the reward function, which discounts the search results of previous search sessions by adjusting the learning rate. We also develop a novel embedding-based Rocchio algorithm to update the queries, which together form an effective solution to the dynamic search problem.

[44] adapts method of [120] (Alpha Zero) to the diverse ranking problem. In this model, they construct a policy-network neural network to generate a policy for Monte Carlo tree search (MCTS) and evaluate the ranking result. The major features of this method include two points. First, it is a listwise method. The system cannot gather the feedback of the users while searching. This method evaluates the whole search and learns from the whole rank generated by MCTS. Second, this model is a self-play model. The LSTM is a strong learner, but MCTS is a weaker one. However, after each ranking, LSTM updates its policy and forwards this to MCTS so that MCTS generates a better result for LSTM to learn. Even though [44] also uses LSTM as an evaluation function in the reinforcement learning framework, they do not utilize some interesting settings of LSTM and augment the dataset like sLSTM. LSTM in [44] is a traditional listwise method, but sLSTM is a stepwise method.

Since most search logs are static in search tasks, building offline interactive search systems is also vital. Previous research used different RL frameworks to model the interactive search process. In cooperation with premium embedding models, RL methods can outperform traditional learning-to-rank (LTR) models [97]. For example,

RLIrank is a policy gradient method with a universal sentence encoder [154, 20]. It outperforms important traditional LTR models like Adarank, ListNet, and RankSVM [144, 142, 22]. Similarly, SlateQ, a Q learning-based method, is proposed to solve the ranking problem in recommendation systems. SlateQ achieves the best performance on RL-based recommendation approaches, outperforming other non-RL approaches, including BERT-based models, demonstrating that DQ has the flexibility to learn a more complex state-action function without worrying about the unstability associated with training a nonlinear function approximator.

For three reasons, reinforcement learning methods are significant in building interactive search systems.

First, reinforcement learning can efficiently model the interactions between users and search agents, allowing feedback to improve search performance.

Second, reinforcement learning can optimize the search performance of the whole search session, reflecting the effect of users' long-term engagement.

Third, reinforcement learning can utilize static logs to train the interactive search models offline, expanding the scope of the data source.

2.2.5 Neural Information Retrieval: Deep Learning for Ranking

Typical neural networks (NN) are constructed by multiple stacked fully connected layers. The features are fed forward and generate output results. In search tasks, the output can be some predicted scores or a vector representing the input. Each layer of the neural network contains a linear transformation of the vector from the previous layer and it is followed by a non-linear activation function. In training process, a loss function measuring deviation of the prediction and the ground truth is minimized by classic back-propagation algorithm [76]. Figure 2.3 shows the architects of NN.

The basic neural networks are further developed with new training mechanisms,

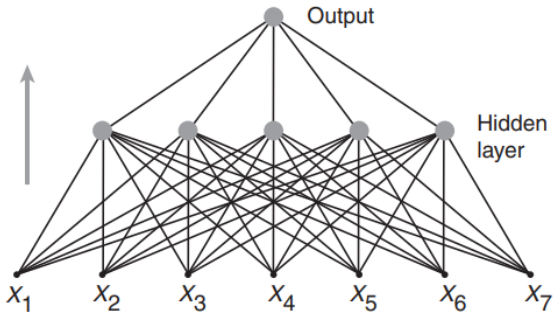

Figure 2.3: Artificial neural networks architecture, from [76]. $X_1, .., X_7$ are the input features.

new sampling approaches and new structures and those models that allows NN structures to be deeper and manage to learn tasks' patterns are called deep learning [76].

Some deep learning models are highly related to search tasks:

- Auto-encoder: An auto-encoder is an unsupervised model learning a data representation. It is trained to construct a lower dimensional representation by setting the input and output the same [136].

- Convolutional neural network (CNN): Unlike densely-connected neural networks, CNN is a partially connected neural network. It involves convolution with kernels to capture local patterns and is followed by a pooling layer to extract important features and reduce dimension [11]. CNN can be used to predict relevance scores.

- Recurrent neural network: A recurrent neural network (RNN) is a sequential model. The input can be some sequential features like sentences. In the search task, sequential features are common. They have different inputs in different time stamps. For example, the list-wise ranking results have different documents in different positions. During the feed-forward process of RNN, the features are extracted and transferred to the next time stamp in each time stamp. With previous features and current input, RNN can generate new features for the next time stamp until the end. Therefore, the final output of the model is associated

with all features at different times [94]. However, during back-propagation of the model, the "vanishing gradient problem" makes the convergence of the models notoriously difficult [58]. To mitigate the issue, LSTM is proposed.

- Long Short Term Memory (LSTM): LSTM is introduced to resolve the gradient vanish problem [123]. In addition to the hidden state vector, LSTM consists of memory cells, including an input gate controlling the influence of new input, a forget gate controlling the influence of previous information, and an output gate controlling the influence of the current hidden state. Those variants allow LSTM to control the gradients better and archive better performance, especially when the input is very long.

Deep learning is one of the most active research areas in recent years [116]. The great success of deep learning in image recognition and natural language processing (NLP) attracts many researchers to explore the possible utilization of deep learning methods in other applications [80][35].

Regarding deep learning in search, recent research focuses on several techniques to improve ranking accuracy, including representing the text content with advanced embedding approaches [38], learning the relevance pattern with large capacity networks [53] and the approaches we discuss in this thesis: capturing users' intent from the interactive search process. Those methods show promising performance on various search benchmarks [66, 157, 154, 156, 155].

More specially, many models of using deep learning for the learning-to-rank problem are proposed. For example, [118] uses a convolution neural network (CNN) to rank short text pairs. [134] proposes a CNN-based method to rank image files. Additionally, [115] applies a neural network to discover the relation for ranking.

Although those methods solve the problem effectively, they do not fit the structure of the interactive search. The experiments in this thesis (Chapter 5 and Chapter 6)

demonstrate that those methods cannot model the interactive search as well as the models with list-wise and interactive structures.

Therefore, some researchers consider some sentence-to-sentence models to model the search. Most of the methods in ranking problems select LSTM or RNN as the 'input gate' of the documents list since they naturally have a sequence structure. This is also the reason we use LSTM. In our work in Chapter 5, we improve the LSTM model to a deep value network by exploring a new learning method.

.

2.2.6 BERT-based ranking systems

Bidirectional Encoder Representations from Transformers BERT is one of the most crucial text content embedding methods [133]. As a pre-trained deep bidirectional transformer, BERT can capture the surrounding contextual signals and provide embedding results based on semantic understanding. In many sequence-to-sequence natural language processing tasks, BERT archives outstanding performance [107]. BERT-based models are an essential part of the models we proposed in this thesis, and some of them are the baselines in the experiments.

In addition to the BERT model, other transformer models are proposed. We briefly introduce some of the important models:

- Transformers: Transformers is an attention-based NN model which can efficiently extract key information from a sequence of data. It consists of encoder and decoder blocks with a softmax activation function. Figure 2.4 shows the structure of the transformer model.

  The input of the model is a sequence of data, for example, a sentence. They are embedded and sent to the positional encoders, which will assign vectors to the data unit, for example, words based on their position. The multi-head attention

and a feed-forward network in the encoder blocks receive the embedding results. The multi-head attention layers generate vectors representing the relationship between each word and others. The feed-forward network transfers those vectors to the decoder block.

The decoder block consists of positional encoders and masked multi-head attention layers. The multi-head attention block takes the attention vectors from the masked multi-head attention layers and encoder block and generates new vectors. Those vectors can represent the meaning of the word in the entire document better. The vectors are further sent to the feed-forward neural network and softmax activation function.

The transformer model is significant in natural language processing (NLP), especially in tasks like machine translation, text classification, and question answering [141].

- Robustly Optimized BERT pre-training Approach (RoBERTa): RoBERTa is a BERT variant with more thorough training. [89]. The training process of RoBERTa uses larger datasets and more delicate hyper-parameters tuning

- DistilBERT: DistillBERT is intended to simplify the structure of BERT with neural network distillation, which can accelerate the algorithm [114]. The distillation learns the complex structure of BERT with a simpler transformer model. The parameter of the new model reduces by 40%. The speed is 60% faster while 97% performance is retrained.

In search, most BERT-based ranking systems are point-wise methods, which means the items are ranked by the similarity scores between the embedding results of the query and the document [104, 99]. Although these methods can utilize the query and document information, they do not learn the users' feedback from the interaction and ignore the ranking bias led by positions, observation, and other reasons

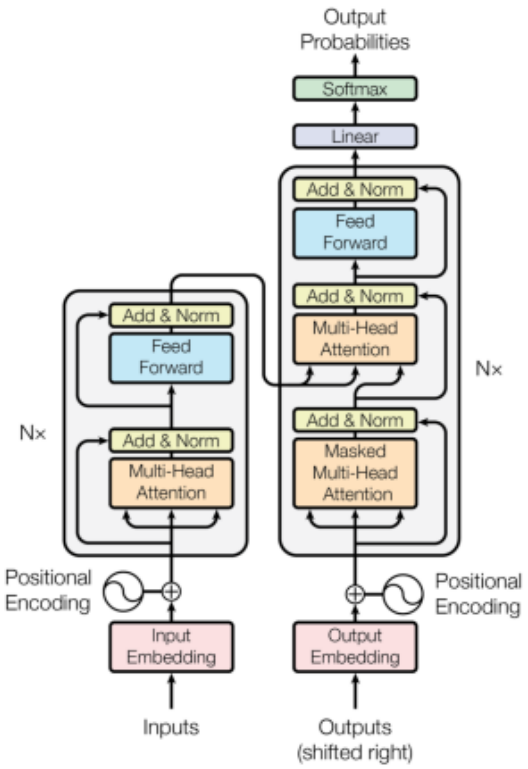

Figure 2.4: Transformer model architecture, from [133].

[46]. Those drawbacks may cause weaker ranking performance than list-wise and interaction-based ranking systems.

2.3 Search Diversification

In this part, we emphasize the importance of evenness, brief the primary baseline method −− Determinantal Point Processes (DPPs), and introduce Simpson's diversity index (SDI) as a diversity model in multi-aspect search problems.

2.3.1 Richness and Evenness

Richness and evenness are essential concepts for understanding diversity. They first develop in biology to measure the diversity of communities. Figure 2.5 is an example of two communities with similar richness but different evenness. Richness reflects the

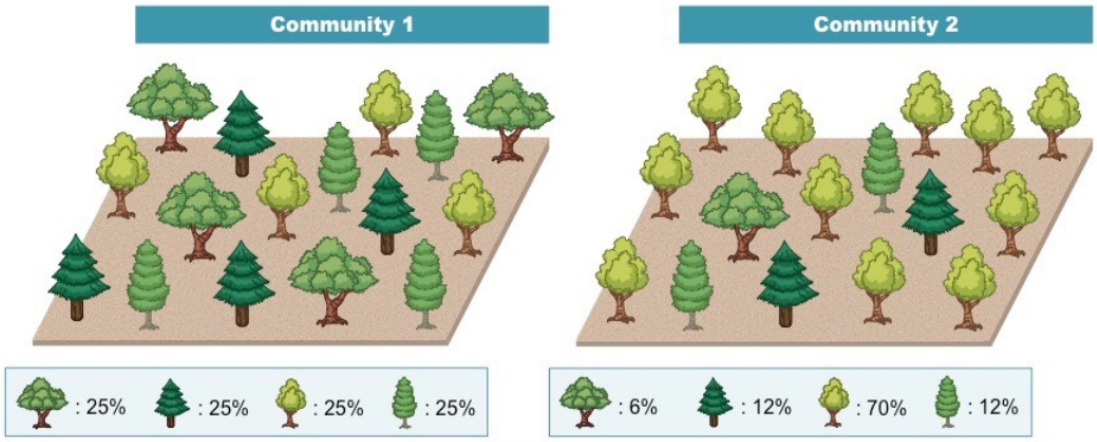

Figure 2.5: Community 1 and Community 2 have similar richness, but Community 2 has higher evenness than Community 2. The figure is from [12].

number of different species, and evenness measures their distribution.

Prior research on search diversification focused on the richness of the results [121]. Richness quantifies the number of different types the dataset of interest contains, while evenness considers the uniformity of the distribution of these types.

However, *evenness* is also vital for search diversification. Significantly for the e-commerce setting, it can affect users' online shopping experience. For example, consider a product catalog with 1000 A items, 500 B items, and 500 C items. Suppose we attempt to retrieve 12 results as diverse as possible to fit on the first page of the results. Without considering evenness, the expected search results might be 6 A items, 3 B items, and 3 C items. While considering evenness, the expected results are all 4. The *richness* of the two versions of the results is identical. However, the information entropy of the second set of results is higher, meaning that higher *evenness* could increase the information that the users receive in a limited space. In a multi-aspect search, evenness is even more important. In multi-aspect search, the search results combine many independent single-aspect searches. To support this conclusion, we can consider a simple two aspects problem. Given two sets A and B, both have Z items. Some of the items belong to a similar class. They have m and n classes. The

number of classes i in A is a_i. The number of classes j in B is b_j. We randomly choose one item from A and B respectively, which can be represented by (x_1, y_1). Then we choose an item from A and B respectively again, which can be presented by (x_2, y_2). We keep doing this operation Z times. We want to know the expected value of the classes of (x_i, y_i). This problem can inspect how evenness influences the richness of a new set combined with two sets their richness is known.

To solve this problem, we can only consider a minor problem. When we select (x, y) and then select (x', y'), what is the probability of $(x, y) = (x', y')$. We can iteratively solve the original problem by keeping solving this minor problem. Therefore, to evaluate the expected value of the new set's richness, we can consider $P((x, y) = (x', y'))$.

$$P((x, y) = (x', y')) = \frac{\sum_{i \leq m, j \leq n} (a_i(a_i - 1)b_i(b_i - 1))}{Z^2(Z-1)^2}$$

By Jensen's inequality, we shows that $P((x, y) = (x', y'))$ has minimal only when $a_i = \frac{Z}{m}$ and $b_i = \frac{Z}{n}$. This result implies that the high evenness of every aspect can promise a high richness of the set with these aspects. [121] further prove that the combinations of the even results are the most diverse in more general situations. Therefore, evenness can affect the richness of multi-aspect searches as well.

The above discussion shows that evenness is significant for diversity analysis. However, the vast majority of the diversity models for search do not consider result evenness. Therefore, we propose to use a model considering evenness to enrich the diversity model for search. We propose to adopt the Simpson's diversity index (SDI) method from biology for this task [121]. We further introduce the DPP-based algorithms for result diversification and then introduce the SDI method.

2.3.2 Determinantal Point Processes for Search Diversification

Determinantal Point Processes (DPPs) are useful in domains where repulsive effects or diversity are important aspects to model. In recommender systems and information retrieval domains, where the discovery of documents and items is significant, DPPs can be used to diversify results.

In search problems, DPPs are a subset selection problem. We define the search space as a set S, whose size is N, and the search results are a subset S', whose size is k. $B_{S'}$ represents the selected items in S'. DPPs can be formulated as follows:

$$X = \underset{S' \subseteq S}{\operatorname{argmax}} \det(B_{S'}^T B_{S'}))$$

X is the volume of a parallelepiped spanned by the vectors of B. It is determined by the vectors' length (relevance) and their cosine similarity (similarity). Figure 2.6 demonstrates how the length of the vector (relevance) and the angles among the vector (similarity) affect the volume X. By maximizing X, we can obtain the best subset of S, considering both diversity and relevance.

However, DPPs are a richness-greedy diversification method because X does not change when we apply linear transformations to revise the evenness of B. Therefore, the evenness of the obtained subset of each aspect is not promised. In a multi-aspect search, the results are a combination of every aspect. As we mentioned in Section 2.3.1, this can lead to weaker diversity of the combination.

2.3.3 Simpson's diversity index

Simpson's Diversity Index (SDI) was originally introduced in biology to measure the evenness and richness of a habitat [121], and aims to estimate the probability of two randomly selected individuals from a sample will be the same. Apparently, this probability is smaller if the habitat has more species (Diversity). Based on Jensen's

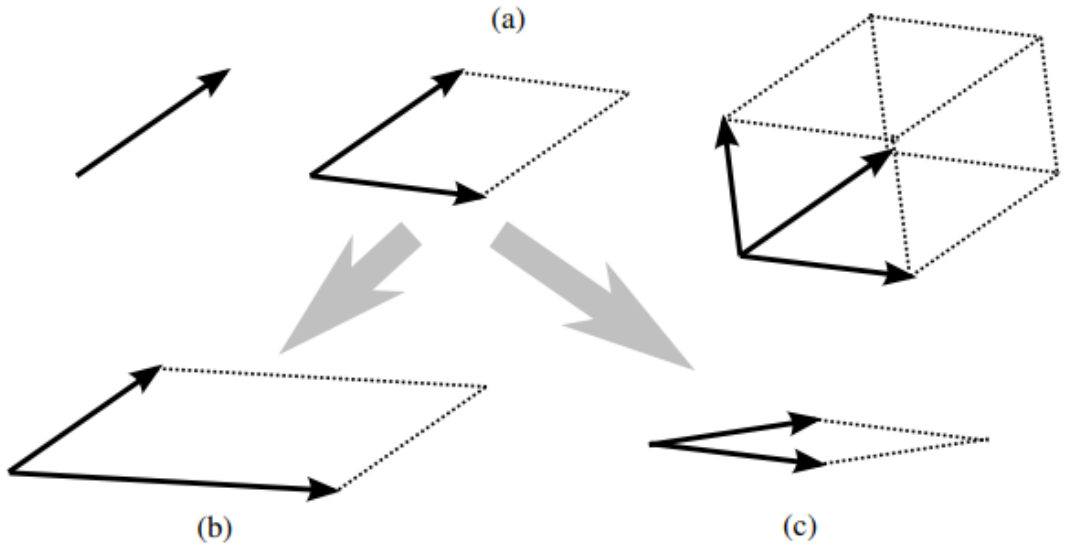

Figure 2.6: A geometry view of DPPs: from (a) to (b), as the magnitude of a relevance vector increases, X also increases. From (a) to (c), as the similarity increases, X decreases. The figure is from [77].

inequality, if the total number of animals in the habitat is constant, this probability is also smaller when the numbers of each species are the same (Evenness).

More formally, SDI is defined as the probability that two items are chosen at random and independently from a set the same:

$$D = \frac{\sum_{i=1}^{R} (n_i(n_i - 1))}{N(N-1)} \tag{2.11}$$

Where R is the number of classes. n_i is the number of items of the ith class. N is the total number of all items. Smaller D means higher diversity.

In Chapter 3, we adapt SDI to the initial search. Compared to the DPPs approaches, the proposed method is faster and more diverse.

2.4 User feedback modeling

Modeling users' feedback is essential for interactive search. The user model digests static user interaction logs and provides users' interest in the search results dynamically, which is a crucial step to dynamically extract the feedback information when various search results are generated by the search system in the interactive search [65]

Users' feedback consists of implicit and explicit feedback. The explicit feedback means labels can directly represent users' interest in the search item. For example, users' ratings, reviews, and 'thumb-up' are explicit feedback. In contrast, implicit feedback is more common in search tasks. For example, users' click, add to the cart, or attention can happen in most search scenarios. However, implicit feedback is more accessible and impacted by some biases, such as observation bias. The observation bias means that the users usually passively browse the search results, so many search items are not observed by the users. Those items can be relevant to the query but are considered irrelevant because they are not observed. Those biases bring much noise to the labels, which can profoundly affect the search tasks.

Previous research extracts that implicit feedback by modeling user behaviors and biases. In this section, we will introduce those methods.

2.4.1 Probabilistic Graphical Models

Most of the traditional click models are based on the PGM framework [75]. In PGM, user interactions are organized as a sequence of events, such as document examinations, skips, and clicks. The most frequent events that those models consider are document examinations and clicks. Since it is instinctively correct that a user's clicks should be led by a document examination, these two events are highly interdependent. Based on this assumption, some important click models, including CCM, DCM, DBN, and UBM, are proposed [41, 51, 24].

2.4.2 Neural Click Models

The neural click model (NCM) is a method based on artificial neural networks. The input data of the artificial neural network is generally one or many vectors, even high dimensional tensor. Therefore, different from the PGM-based method, NCM represents the clicking process with vectors [13]. [13] attempts to use many different neural networks to construct the click models, including Long Short-Term Memory (LSTM) [60] and recurrent neural networks (RNN) [149]. Their empirical evaluations showed that the LSTM version has better performance.

Additionally, NCM also expands to some complicated click models. For example, [26] develops a session search click model (CACM), which leverages the signals from previous search iterations to predict the next search iteration of the search session.

2.4.3 User Bias Modeling

In search, users' click behavior is significant to capture their intent and determine their purchase. An effective click simulator can evaluate and improve the search ranker with less effort than actual users. To support the click simulator design, many techniques are proposed and developed. In this section, we introduce and comment on some important and recent techniques for designing a click model, which include Counterfactual Learning (CL), the user browsing model (UBM), and Reinforcement Learning to Rank (RLIRank). In applications like recommender systems and interactive searching, ranking involves a more complicated information flow. Compared to end-to-end structures, reinforcement learning can organize the information flow better. Additionally, reinforcement learning can incorporate partially observed settings of the click dataset, which may reduce noise from the not observed data. For reinforcement learning, the most crucial issue is how we fit searching settings into

reinforcement structures. We use an example to explain how reinforcement learning to rank works.

A user click model is a set of rules simulating user click behavior on a search engine result page [32]. Some rules are commonly aligned with most of the users. For instance, search engine users pay more attention to the top-ranking document, which is called the position bias effect [25]. Another example is novelty bias. Some research shows that unseen documents easily attract users [151]. Those rules can still roughly capture users' click patterns by ignoring users' personalities or session behaviors. However, the click model built by those rules cannot provide targeted services based on users' preferences. It can be further improved in the e-commerce search scenario, rich with users' click history and interaction. For e-commerce search users, it is common for them to have a profile depicting their purchasing habits. Besides, their search session behaviors, such as queries' revision, can provide some clues about the user's intent. Those patterns help predict their possible click or purchase. Consequently, an effective users' click model in e-commerce should consider general and personal patterns.

Unlike learning to rank, a click simulator does not generate a list of ranked documents. Instead, a click simulator simulates users' reactions to the search results, including click, browse, leave and purchase. To describe the possibility of those reactions, many metrics are developed. For instance, many researchers consider that users are more likely to click more relevant documents, so they use a relevant metric – NDCG, to evaluate the quality of the search results [137]. However, the connection between the reactions and the metrics is indirect. They usually cannot be explained by a particular metric. Many reasons may lead to a user's reaction. For instance, the user clicks the document, maybe because it is relevant or novel. Therefore, optimizing the learning-to-rank model should consider many goals at the same time [17].

Notwithstanding, a click simulator can avoid the aforementioned challenge. As-

suming the click simulator can effectively simulate users' reactions, we can set the optimization goal as a particular reaction, such as purchasing. It would be easier to train learning to rank model for a particular purpose or many of them.

Another advantage of a click simulator is that it allows the training process of learning to rank (LTR) to be more data-driven. It is notorious that the LTR problem is hard to utilize the data directly. While training the LTR, the model generates typically a new list of documents which does not exist in the data set. It is impossible to evaluate such search results accurately based on the data set. To resolve such an issue, we have to apply Hamming distance or interleave technologies to approximately evaluate the search results [31]. Another possible resolution is assuming a score is helpful to improve the performance of the LTR, such as BM25 [110]. Then, based on the data set, we can calculate a score for each pair of queries and document and rank them based on the score. However, those indirect methods cannot evaluate the search results in a list-wise way. The interleaving results are a new list of search results. The BM25 is a point-wise score, which ignores the position of the document in the search results. In real-life applications, users may react differently when the same document is in different positions.

The third challenge of modeling click is the dataset itself. The click dataset is the implicit dataset [100]. In most LTR datasets, the relationship between the labels and the features is explicit. We usually have a score to describe the relevance, diversity, and other properties, so we assume every item in the dataset is observed. However, in click, we only know that some results are clicked. If the items are clicked, it is observed. Regarding the item not clicking, we do not know whether they are observed. Therefore, those data are more like noise for the training.

To cope with the aforementioned challenges, many models are proposed to simulate the click. Basically, we can conclude them as three types:

- Feature-based click simulator.

- Counterfactual learning to rank.
- Reinforcement learning to rank.

Featured-based click simulators predict the users' click behaviors based on the features of the query and the documents. [36] proposes a cascade model (CM), which assumes the user scans documents on a page from top to bottom until a relevant document is found. This can only work for the most straightforward situation where the user keeps browsing the search results and only clicks once. However, users sometimes may click on many documents. To handle this kind of situation, [52] [51] extends CM to the dependent click model (DCM) and click chain model (CCM), which can predict the next click based on a session of previous clicks in a query session. With similar assumptions, [23] also proposes the dynamic Bayesian network model (DBN). All those methods assume the users' click is only affected by the actions in a query session. [32] concludes them as the basic click models. Advanced click models may consider more complex scenarios, such as searching with multiple queries or incorporating users. For the task with multiple queries, [152] proposes the task-centric click model (TCM). TCM is a general model considering query and duplicate bias, which can be built on top of any basic click model. Additionally, [152] further adds user-based parameters into a click model to capture users' information. Those models consider two points [100]:

- Estimating the attractiveness of the documents (relevance).
- Considering other factors that cause or prevent clicks (biases).

Therefore, those methods generally are feature-based evaluations. We use a classic feature-based model, the user browsing model (UBM), as an example to analyze this type of method.

Except for feature-based evaluation, the click of the search results can also be simulated by frequency-based methods. [1] proposes a frequency-based evaluation, which

is counterfactual learning to rank (CLTR). Unlike the feature-based approach, CLTR does not directly predict the user's reaction. Instead, it tries to give an unbiased estimation of the click dataset. When the LTR model generates a new list of results, CLTR simulates the click by analyzing the frequencies of those query-document pairs in the dataset. Although those pairs may be in different positions or appear with other documents, the unbiased estimation can ease the influence of those biased factors. CLTR cannot directly predict users' clicks but provides an unbiased dataset to the LTR model. The experiments of [1] reveal that it can improve the performance of SVM or end-to-end neural networks based on some addictive metrics, like NDCG.

The last method simulating the click is reinforcement learning to rank [154]. The basic idea of this type of method is modeling the whole search process and considering the user's click as the Reward. The goodness of this method is flexibility. Since we can design every step of search in reinforcement learning, it can be adapted to fit a variety of searching, like online search, dynamic search, and session search. Since the user's clicks are considered the Reward, we avoid the influence of various biases and focus on the actions that can improve the click.

In the following sections, I demonstrate three important methods of modeling click for the learning to rank, which is the user browsing model (UBM), counterfactual learning to rank (CLTR), and reinforcement learning to rank (RLR).

2.4.4 User Browsing Model

The user browsing method is a Bernoulli-based model. It predicts user clicks by modeling related factors, such as distance, ranking, and examination. Those factors can be estimated by the likelihood maximization model.

The naive version of the UBM is based on the following situation: A user issues a query and obtains a list of documents from the search engine. The user browses the results from the first one to the last one. For each document, the user first decides to

read the brief or not. If the brief is attractive enough, they click on the document. After that, they continue browsing the following brief.

Let us denote the position of the document as r. We use a binary random variable e to represent whether the user examines the brief. The authors assume that the probability of the examination is dependent on the distance d from the last click (Assuming position 0 as the first click) and the document's position r, because the users tend to abandon the search after a long sequence of unattractive documents. The conditional probability can be further described by Bernoulli distribution:

$$P(a|u,q) = \alpha_{uq}^{a}(1-\alpha_{uq})^{1-a} \tag{2.12}$$

$$P(e|r,d) = \gamma_{rd}^{e}(1-\gamma_{rd})^{1-e} \tag{2.13}$$

Where α_{uq} is the probability of attractiveness of the brief u when the query is q. γ_{rd} is the probability of examination when the distance is d and position is r. a is the attractiveness, which is a binary parameter. e is whether the document is examined.

The full model is a joint distribution of Equation 2.12 and Equation 2.13:

$$P(c,a,e|u,q,d,r) = P(c|a,e)P(e|d,r)P(a|u,q) = P(c|a,e)\gamma_{rd}^{e}(1-\gamma_{rd})^{1-e}\alpha_{uq}^{a}(1-\alpha_{uq})^{1-a} \tag{2.14}$$

Where c is whether the document is clicked. To compute this, different situations should be considered. If we observe a click ($c = 1$), it is obvious that the brief is attractive and the document is examined, which means $a = 1$ and $e = 1$. Therefore, Equation 2.14 can be simplified as:

$$P(c = 1|u,q,r,d) = \gamma_{rd}\alpha_{uq} \tag{2.15}$$

Let us marginalize over a and e in Equation 2.15 when $c = 0$:

$$P(c = 0|u, q, r, d) = 1 - \gamma_{rd}\alpha_{uq} \tag{2.16}$$

To estimate the parameters α and γ, the likelihood maximization method is conducted. It is assumed that each click observation is independent. We can separate the whole set as S^1 and S^0. In S^1, the clicks are observed, while S^0 is its complement. The probability of the observation can be written as:

$$P(Obs|\alpha, \gamma) = \prod_{r=1}^{R}\prod_{d=1}^{D}\prod_{(u,q)\in S^1_{rd}} \gamma_{rd}\alpha_{uq} \prod_{(u,q)\in S^0_{rd}} (1 - \gamma_{rd}\alpha_{uq}) \tag{2.17}$$

The parameters α and γ can be estimated by maximizing Equation 2.17.

Overall, UBM is a Bernoulli-based method and uses the likelihood maximization method to estimate the parameters that the authors consider important to determine the click.

2.4.5 Counterfactual Learning to Rank

Predicting users' click behavior suffers from many biases, such as position, ranking, and diversity. CLTR is proposed to give an unbiased estimation of users' click behavior and predict the users' future clicks.

The key technique of counterfactual learning is to incorporate the propensity of obtaining a specific example into an Empirical Risk Minimization (ERM) object, which is unbiased. In CLTR, this technique is applied to build the nonlinear model with additive IR metrics, such as NDCG and Precision@k.

Unbiased Estimation of Rank-based IR Metrics

To illustrate how counterfactual learning generates an ERM, we first consider the classic structure of the additive ranking performance metrics:

$$\Delta(Y|x_i, r_i) = \sum_{y \in Y} \lambda(rank(y|Y)) r_i(y) \tag{2.18}$$

Where Y is a ranking of results, x_i is a query instance. $\lambda()$ can be any weighting function that reflects the discount of the ranking position. $rank(y|Y)$ is the ranking position of y in Y. $r_i(y)$ denotes the user-specific relevance of instance x_i.

Apparently, different $\lambda()$ functions can result in different metrics. For instance, when $\lambda(rank) = rank$, it gives the sum of relevant ranks metric. However, when $\lambda(rank) = \frac{-1}{log(1+rank)}$, it gives the DCG metric.

To estimate the performance of a system when a ranking list Y is given without biases, we can go through all the probabilities of every document and their relevance score.

$$R(S) = \int \Delta(S(x)|x, r) dP(x, r). \tag{2.19}$$

Where S is a ranking system that returns the ranking Y when the user gives a query x_i. $P(x, r)$ is the frequency of x with relevance score r.

To apply Equation 2.19 to the actual dataset, we need to point out an important difficulty. When dealing with implicit feedback data like clicks, it is impossible to observe all relevance r_i. Particularly, a click can be noise and indicate the positive relevance of the presented documents. At the same time, a missing click may be led by the ignorance of the users, which does not necessarily indicate weak relevance. Consequently, $R(x)$ can only be estimated under a partial-information setting. We can use a 0/1 vector $o_i \sim P(o, x_i, \bar{Y}_i, r_i)$ to indicate which relevance values are revealed.

In particular, we can define the distribution of o_i as $Q(o_i(y)|x_i, \bar{Y}_i, r_i)$, given a query instance x_i and presented ranking $\bar{Y}$. With this counterfactual setting, we can further estimate $\Delta(Y|x_i, r_i)$:

$$\hat{\Delta}_{IPS}(Y|x_i, \bar{Y}_i, o_i) = \sum_{y:o_i(y)=1 \bigwedge r_i(y)=1 \bigwedge y \in Y} \frac{\lambda(rank(y|Y))}{Q(o_i(y) = 1|x_i, \bar{Y}_i, r_i)} \tag{2.20}$$

Equation 2.20 is unbiased, because,

$\mathbb{E}_{o_i}[\hat{\Delta}_{IPS}(Y|x_i, \bar{Y}_i, o_i)]$

$= \mathbb{E}_{o_i}\left[\sum_{y:o_i(y)=1 \bigwedge y \in Y} \frac{\lambda(rank(y|Y))r_i(y)}{Q(o_i(y)=1|x_i,\bar{Y}_i,r_i)}\right]$

$= \sum_{y \in Y} \mathbb{E}_{o_i}\left[\frac{o_i(y)\lambda(rank(y|Y))r_i(y)}{Q(o_i(y)=1|x_i,\bar{Y}_i,r_i)}\right]$

$= \sum_{y \in Y} \lambda(rank(y|Y))r_i(y)$

$= \Delta(Y|x_i, r_i).$

We assume $Q(o_i(y) = 1|x_i, \bar{Y}_i, r_i) > 0$ when all y are relevant. Equation 9 shows that the IPS can be estimated without relevance or missing observation.

Based on Equation 2.20 [128], we can use IPS weighting to give an unbiased estimation of $R(S)$.

$$\hat{R}_{IPS}(S) = \frac{1}{N}\sum_{i=1}^{N} \sum_{y:o_i(y)=1_i(y)=1} \frac{\lambda(rank(y|S(x_i)))}{Q(o_i(y) = 1|x_i, \bar{y}_i, r_i)}. \tag{2.21}$$

Equation 2.21 is also called the risk of a ranking system S. What is important, the propensities $Q(o_i(y) = 1|x_i, \bar{y}_i, r_i)$ are unknown. Normally, we should apply some methods to estimate this distribution. For example, [3] designs a distance model. Except for this, many practical models are also proposed [43][69][135].

In summary, counterfactual evaluation models click on the users' propensity and estimate the system's performance with an unbiased setting. Therefore, even though the ranker may generalize results that do not exist in the log, CL can still evaluate the quality of the ranker and model the click results. For missing data, the CL has the same effect as "pooling" in machine learning, which can estimate the missing points based on the existing neighborhood points.

2.4.6 User Modeling Evaluation

When modeling the click, three aspects are of vital importance, which are scalability, unbiasedness, and flexibility. This part discusses the aforementioned methods based

on these three aspects.

Scalability

Scalability is crucial for a click model because the volume of click logs is generally huge. For example, the number of click logs of a popular e-commerce website, like The Home Depot, can reach millions a day. To analyze those data and generate helpful feedback in time, a click simulator must process them in a reasonable time.

The scalability of UBM is extraordinary because its computation complexity is $O(N)$, N is the number of click items. Estimating two critical parameters α and γ is essential. However, with the likelihood maximization method, the estimation cost as much as Bayesian inference.

In terms of CLTR, to reduce the bias of a pair of queries and items, we have to search for all these kinds of pairs in the click logs first. Therefore, if we have M different pairs, N click logs, the complexity of the search is $O(MN)$. After we find all the relevant logs, it only costs $O(n^2)$ to compute the IPS score, n is the number of this kind of pair. This reveals that the scalability of CLTR is worse than UBM.

For RLR, the click model must go through the search logs, whose time complexity is only $O(N)$. However, the selection of the reward function is essential, because the model has to train the reward function after each search session. If the reward function is a neural network, the algorithm can be very time costly.

Therefore, the scalability of UBM is the best among these three methods. CLTR is weaker but predictable. The scalability of RLR depends on its reward function.

Unbiasedness

The click logs dataset is an implicit dataset. The bias of the click can lead to many noises, degrading the training models' performance. Therefore, it is necessary to consider unbiasedness when modeling the click.

UBM assumes that the bias of the click or the possibility of the item being observed is related to the distance between the clicked item and the item to estimate. It is unclear whether the assumption is fundamentally correct, but adjusting some parameters can somehow reduce the bias of the items. The experiment also supports that UBM performs better than the model without considering the bias issue.

CLTR constructs a counterfactual model to estimate the bias of the click. Given a metric we want to optimize, CLTR can construct an unbiased, theoretically correct estimation. Therefore, CLTR promises the minimum of the model's bias.

The unbiasedness of RLR depends on how you construct the model. If we consider the observation step and assign a reward function to this part, we can reduce the bias well. It does not promise that the bias is minimal, but we can consider more metrics in this process instead of only considering one in CLTR. Therefore, RLR can have some mechanisms to promise unbiasedness.

Flexibility

Flexibility is essential because the click scenarios are varied. In real life, different search engines generate different search logs. For example, in dynamic searching, we may have an interaction process. Therefore, the click model should be flexible.

UBM needs to be more flexible. The basic assumption of UBM is that the items are on pages, and each item has a brief. To utilize UBM, the click logs should be organized in this form, which is difficult or impossible for some search scenarios.

CLTR is a method to restrict bias, so it does not have an optimizer. Therefore, the selection of CLTR is flexible. However, CLTR can not purely be defined by the click. It requires some scores to compute the IPS. For example, when you select NDCG as $\lambda()$, your dataset should have the relevance score between the query and the document. Therefore, CLTR has to select a proper $\lambda()$ based on the dataset.

RLR is the most flexible method. Reinforcement learning improves the Agent by

processing the feedback. In click logs, the users' clicks are the feedback. Therefore, when modeling the click, RLR can always properly define its components, such as Reward, Agent State, and Action. Additionally, RLR can be reconstructed in different ways based on your search scenarios. Therefore, RLR is the most flexible method.

Conclusions

In this section, we discuss the essential background of click modeling and three commonly used and important methods, which are UBM, CLTR, and RLR. We consider the three most important aspects when we discuss these three methods, including scalability, unbiasedness, and flexibility. It reveals that the drawbacks and advantages of those methods are apparent. UBM's scalability is the best, but its flexibility is weak. It also has some mechanisms to handle bias based on naive assumptions. CLTR can theoretically erase the bias, but its scalability is weaker than UBM and is less flexible than RLR. RLR is the most flexible method, but many aspects of RLR have to be defined based on the search scenario. It is also essential to consider the scalability issue when using RLR.

In the future, click simulators will still be an important topic in learning to rank and search. The cooperation of all those three methods can generate some fascinating methods. For example, the IPS of CLTR can be the heuristic function for UBM. This survey is useful for future studies.

Chapter 3

An Efficient Approach to Search Diversification

Initial search results are the base of interactive search. In this section, I extend my previous work [156] that published in SIGIR to introduce an efficient approach for search diversification.

In search and recommendation, diversifying the multi-aspect search results could help with reducing redundancy, and promoting results that might not be shown otherwise. Many previous methods have been proposed for this task. However, previous methods do not explicitly consider the uniformity of the number of the items' classes, or *evenness*, which could degrade the search and recommendation quality. To address this problem, we introduce a novel method by adapting the Simpson's Diversity Index from biology, which enables a more effective and efficient quadratic search result diversification algorithm. We also extend the method to balance the diversity between multiple aspects through weighted factors and further improve computational complexity by developing a fast approximation algorithm. We demonstrate the feasibility of the proposed method using the openly available Kaggle shoes competition dataset. Our experimental results show that our approach outperforms previous state of the

art diversification methods, while reducing computational complexity.

3.1 Diversifying Multi-aspect Search Results Using SDI

In this section, we present a method that diversifies multi-aspect search results using Simpson's diversity index. We first present an important variation of Simpson's diversity index and then explain how we apply it to search. We conclude by constructing a quadratic program to compute SDI more efficiently.

3.1.1 Diversifying Multi-aspect Search Results using Simpson's diversity index

In multi-aspect search, items are labeled by many aspects. We first consider the SDI of an aspect. In addition to Equation 2.11 in Section 2.3, which is:

$$D = \frac{\sum_{i=1}^{R}(n_i(n_i-1))}{N(N-1)} \tag{3.1}$$

We have another way to calculate SDI.

In a set, we denote the number of items in this set as N. R is the number of items' classes. n_i is the number of items of the ith class.

When we randomly select two items as a pair, there are $\frac{1}{2}N(N-1)$ pairs. For each pair, they belong to the same class or not. The total number of pairs belonging to the same class is $\sum_{i<j\leq N}(a_i \& a_j)$. Where a_i is the Item i. If two items are the same $(a_i \& a_j) = 1$, else $(a_i \& a_j) = 0$. If we also count the item itself, we have:

$$D(p, S') = \frac{2\sum_{i,j\in S', i\leq j}(a_{p,i} \& a_{p,j})}{|S'|(|S'|+1)} \tag{3.2}$$

Where p is the aspect, P is the aspect set, S' is the selected set, S is the whole set, $|S|$ is the size of S. $a_{p,i}$ is the Aspect p of Item i.

If we consider all the aspects at the same time, we have:

$$H(S') = \sum_{p \in P} \omega_p \frac{D(p,S')}{D(p,S)}$$

In this equation, we first normalize each aspect's SDI, because different aspects' SDI may have different scales. Then, we weigh every aspect with ω_p. If we only consider diversification, $\omega_p = 1$. However, in the search problem, relevance is also very important. When the item is highly relevant to some aspects. For example, when users search 'blue shoe', it is not necessary to present other colors' shoes anymore. Therefore, the 'color' aspect should be penalized. We let $\omega_p = 1 - \frac{k_p - 1}{|S'| - 1}$, Where k_p is the number of the most common aspect of Aspect p in top $|S'|$ relevant results. For convenience, we let $\phi_p = \frac{\omega_p}{D(p,S)}$.

This transformation of SDI helps us organize it to a more efficient form to be optimized $--$ the binary quadratic program (BQP).

3.1.2 The Binary Quadratic Program

In this section, we transfer the SDI optimization to a BQP problem, which has thorough previous research to systematically be approximated and optimized [91][34][122]. Since $H(S')$ is pairwise, it can be computed in an easy way. Let's expand $H(S')$.

$$H(S') = \sum_{i,j \in S', i \leq j} \frac{2}{|S'|(|S'|+1)} \sum_{p \in P} \phi_p(a_{p,i} \& a_{p,j})$$

Let $Q_{i,j} = \frac{2}{|S'|(|S'|+1)} \sum_{p \in P} \phi_p(a_{p,i} \& a_{p,j})$, we can reformulate $H(S')$ as $H(S') = \frac{1}{2} x^T Q x$, x is a vector representing S'. The length of x equals to $|S|$. Each entry is an item of S. The entry of the chosen item is 1, or it is 0. If we further consider the relevance, we have:

$$\min_x T(S') = \min_x H(S') + R(S') = \min_x \frac{1}{2} x^T Q x + b^T x \tag{3.3}$$

Where b is the relevance vector. Consequently, it becomes a binary quadratic program problem, whose constraints are: s.t. $x_i \in \{0, 1\}, \sum_{k=1}^{|S|} x_k = |S'|$, where x_i is the entry of x.

Equation 3.3 is a classical BQP problem. In this work, we first relax it to a convex optimization problem [91]. Then we use the Frank–Wolfe algorithm to optimize it [34]. Finally, we use Goemans and Williamson procedure for the approximate solutions [122]. Such a procedure guarantees that the solution is a close approximation that $T \in [T_{min}, \frac{\pi}{2} T_{min}]$.

3.2 Experimental Setting and Results

We first introduce the experimental datasets, including the queries generation and items' collection. Then, we describe the evaluation methods, initial relevance ranking, and the baseline methods used to report the experimental results.

3.2.1 Datasets

We experiment with two datasets: a publicly available Kaggle benchmark for the searching items dataset, and a complementary dataset constructed using publicly available Amazon Web Search engine interface for the queries dataset.

Kaggle Shoes Price Competition Dataset (Kaggle): Kaggle shoe price dataset is a list of shoes and their associated information, containing 19046 women shoes and 19387 men shoes. However, the items of this dataset also contain clothes, gloves, jewelry, and so on. Some items do not have features. To focus on the diversity problem, we remove items without features, clothes items, jewelry items, gloves items, trousers items, and toys items. At last, we have 14609 items. Besides, even though the items have many aspects, they do not have the values of many aspects. Therefore, we only choose 21 most commonly used features: 'Season', 'Material', 'Gender', 'Shoe

Size', 'Color', 'Brand', 'Age Group', 'Heel Height', 'Fabric Material', 'Shoe Width', 'Occasion', 'Shoe Category', 'Casual Dress Shoe Style', 'Shoe Closure', 'Assembled Product Dimensions (L x W x H)', 'Fabric Content', 'Shipping Weight (in pounds)', 'prices.offer', 'prices.amountMin', 'prices.amountMax' and 'prices.isSale'.

Amazon Queries Suggestions Dataset (AQS): To complement the Kaggle queries dataset, we created a new dataset automatically constructed for the "shoes" category using the query suggestions provided by the Amazon Web Search engine. Specifically, we collected the queries suggestions for the top-level category 'shoes'. This process resulted in 373 queries, such as 'fur lined winter coat women' and 'leather boots for men'. The AQS dataset is available at the URL `https://docs.google.com/spreadsheets/d/17DMU5pMSiNxi05yu5pdEvrUf3b0mIAF1rYn2ypruS7A/edit#gid=0`.

3.2.2 Baselines and Experimental Setting

Setting

In the experiments, we search for queries and every model returns the top 10 results. We used the averaged results of all queries as the data points.

All the diversification methods considered, and our proposed method, require the relevance vectors and the similarity matrices. In this work, BM25[110] score of each item with respect to the query. For the item-item similarity matrix, the number of identical aspects between the items is used, which could be improved in future work.

To trade off the relevance and diversity, we introduce a parameter θ. θ adjusts the relation between diversity and relevance in the following way:$(1-\theta)H(S') + \theta R(S')$. For the proposed method, the approximate rate ϵ is 0.0001.

Baselines

We use three baselines, including Greedy version DPPs and two recent and influential variations. **1) DPPs Greedy:** Greedily selecting items to maximize the determinant [14]. **2) k-DPPs:** It is a sample-based DPPs, which can accelerate the Greedy DPPs. However, since it localizes the optimization, the performance may be unstable [78]. **3) Fast MAP DPPs:** A novel approach improves the computation complexity of the maximum a posteriori (MAP) inference DPPs, which is the state of art DPPs algorithm [28].

3.2.3 Evaluation Metrics

We now summarize the metrics used to compare the methods.

- **Coverage Rate(CR):** The coverage rate measures the diversity of the subset [19]. For each aspect of the item, they have at most 10 features or the number of this aspect's features in this set. The coverage rate is a diversity metric.

  $$CR(S') = \frac{1}{|P|} \sum_{p \in P} \frac{|A'_p|}{\min(|S'|, |A_p|)}$$

  Where S' is the selected set, P is the set of all aspects, A'_p is the features of Aspect p in the selected set and A_p is the features of Aspect p .

- **NDCG@10:** We use normalized discounted cumulative gain (NDCG) to measure ranking results of the relevance. Since both results of DPPs and the proposed method do not have the order, we rank the selected subset based on the items' relevance before computing the NDCG@10 [19].

- **α-NDCG:** A variations of NDCG, which can measure both relevance and diversity [19].

- **Variance:** We use averaged variance of every aspect's number of features to measure the evenness [109].

$$V(S') = \frac{1}{|P|} \sum_{p \in P} \sqrt{\frac{\sum_{a \in p} (|a| - \mu_p)^2}{|p|}}$$

Where μ_p is the averaged same features number of Aspect p in subset S'. a is the feature of Aspect p. $|a|$ is the number of this feature in the subset.

3.2.4 Results

The results of diversity, evenness, and relevance are presented in Figure 3.1, 3.2, 3.3, 3.4. The result of the computing time is illustrated in Table 1. In 3.1, 3.2, 3.3, 3.4 the trade-off parameter θ ranges from 0 to 1, whose step size is 0.1. For CR(3.1), NDCG@10(3.2), and α-NDCG(3.3), the larger their value, the better the performance of the model is. For variance(3.4), a lower line means the model is evener. In Table 1, less computing time means the model is faster. Generally, the proposed method outperforms the baselines in terms of relevance, diversity, and evenness, especially when θ ranges from 0.3 to 0.7. The improvement is up to more than 15%. In terms of computing time, the proposed method spends less time than most of the baselines.

We use CR to measure the richness of the multi-aspect results in Figure 3.1, which shows the proposed method outperforms the baselines when θ ranges from 0.2 to 0.8. In terms of relevance, the NDCG@10 scores from Figure 3.2 show that the proposed method has closed performances. We further use α-NDCG to measure the relevance and diversity at the same time in Figure 3.3, which demonstrates that the proposed approach has the best performance while θ ranges from 0.4 to 0.7. Figure 3.4 further applies the Variance to measure the evenness of the returned results. In this figure, we find that when the model emphasizes diversity (θ closes to 0), the variance of SDI is smaller than all the baselines. We further test the computing time of the proposed method and summarize the results in Table 1, revealing that the proposed method is slower than k-DPPs but faster than other baselines.

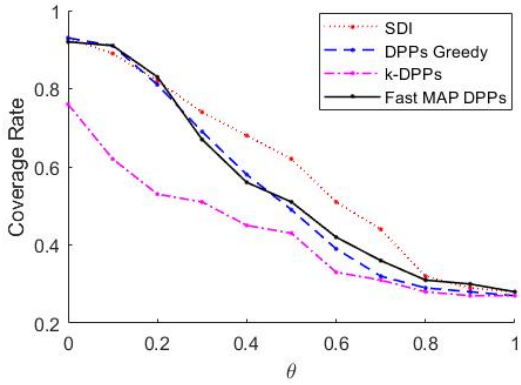

Figure 3.1: Comparison of trade-off coverage rate performance between relevance and diversity under different choices of trade-off parameters θ on Amazon Queries Suggestions dataset and Kaggle Shoes Price Competition Dataset.

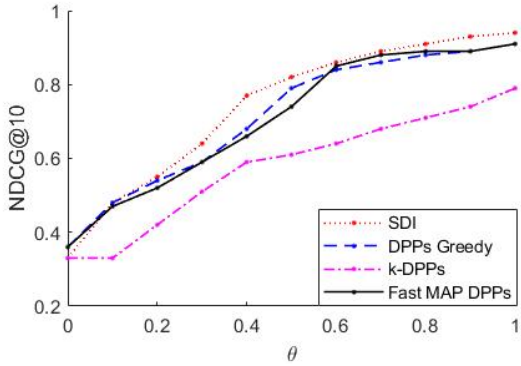

Figure 3.2: Comparison of trade-off ndcg performance between relevance and diversity under different choices of trade-off parameters θ on Amazon Queries Suggestions dataset and Kaggle Shoes Price Competition Dataset.

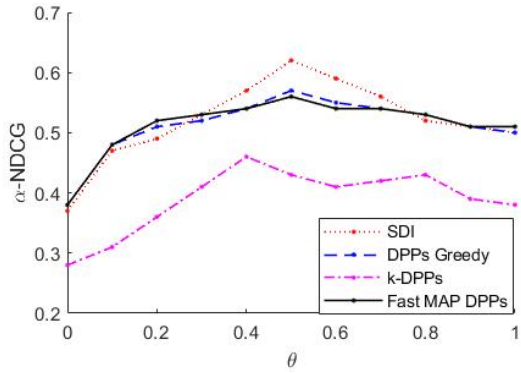

Figure 3.3: Comparison of α-NDCG performance between relevance and diversity under different choices of trade-off parameters θ on Amazon Queries Suggestions dataset and Kaggle Shoes Price Competition Dataset.

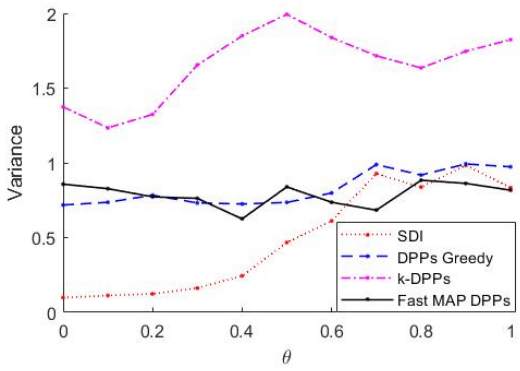

Figure 3.4: Comparison of trade-off variance performance between relevance and diversity under different choices of trade-off parameters θ on Amazon Queries Suggestions dataset and Kaggle Shoes Price Competition Dataset.

Table 3.1: Comparison of average running time (in milliseconds)

Model	SDI	DPPs-Greedy	k-DPPs	Fast-MAP-DPPs
Computing Time	45.32	1093.48	17.86	98.45

3.3 Analysis & Discussion

We now discuss the proposed method's advantages and drawbacks based on the experimental results above. Figure 3.1, 3.2, 3.3 show that our SDI method outperforms all the baselines while we consider diversity and relevance with similar or closed weights. Simultaneously, Figure 3.4 shows that the evenness of SDI is substantially higher than the baselines', supporting the analysis in Section 2.

Table 1 reports the computation time of 4 methods, which aligns with their computation complexity. Greedy DPPs' is $O(N^4)$. Fast-MAP-DPPs' is closed to $O(N^3)$. k-DPPs' is $O(Nk^2)$. The proposed method has $O(1/\epsilon)$ iterations, and each iteration is just linear programming. Although the proposed method is slower than k-DPPs, k-DPPs have the worst general performance in Figure 3.1, 3.2, 3.3, 3.4, which still supports the conclusion that SDI is a competitive diversity model.

Chapter 4

De-biased Modeling of User Feedback

Modeling users' feedback is a key step toward interactive search. In this section, I extend my previous work [157] that published in CIKM to demonstrate a de-bias method for the users' click, which is one of the most important feedback we receive from the search.

Users' clicks on Web search results are one of the key signals for evaluating and improving web search quality, and have been widely used as part of current state-of-the-art Learning-To-Rank(LTR) models. With a large volume of search logs available for major search engines, effective models of searcher click behavior have emerged to evaluate and train LTR models. However, when modeling the users' click behavior, considering the bias of the behavior is imperative. In particular, when a search result is not clicked, it is not necessarily chosen as not relevant by the user, but instead could have been simply missed, especially for lower-ranked results. These kinds of biases in the click log data can be incorporated into the click models, propagating the errors to the resulting LTR ranking models or evaluation metrics. In this work, we propose the De-biased Reinforcement Learning Click model (DRLC). The DRLC

model relaxes previously made assumptions about the users' examination behavior and resulting latent states. To implement the DRLC model, convolutional neural networks are used as the value networks for reinforcement learning, trained to learn a policy to reduce bias in the click logs. To demonstrate the effectiveness of the DRLC model, we first compare performance with the previous state-of-art approaches using established click prediction metrics, including log-likelihood and perplexity. We further show that DRLC also leads to improvements in ranking performance. Our experiments demonstrate the effectiveness of the DRLC model in learning to reduce bias in click logs, leading to improved modeling performance and showing the potential for using DRLC for improving Web search quality.

4.1 De-Biased Modeling of Search Click Behavior with Reinforcement Learning

In this part, we introduce the proposed DRLC model. We first introduce the implementation of the DRLC's components, namely the value networks, which include a bias network, and a de-biased network. Then, we overview the proposed reinforcement learning (RL) approach to training a de-biased click model.

4.1.1 Value Network Implementation

We chose to use Convolutional Neural Networks (CNNs) as our value networks. The first CNN C_1 is a bias network, which means this network considers the observation bias: Some documents are not clicked because they are not observed. The input features are bias features B and the document features D. B is a vector with 1 or 0. B represents the observation situation of the search results. If the item is observed, the value is 1 or it is 0. We use an observation window to observe the SERP. We first assume the documents browsed by the window are observed. Then, by comparing the

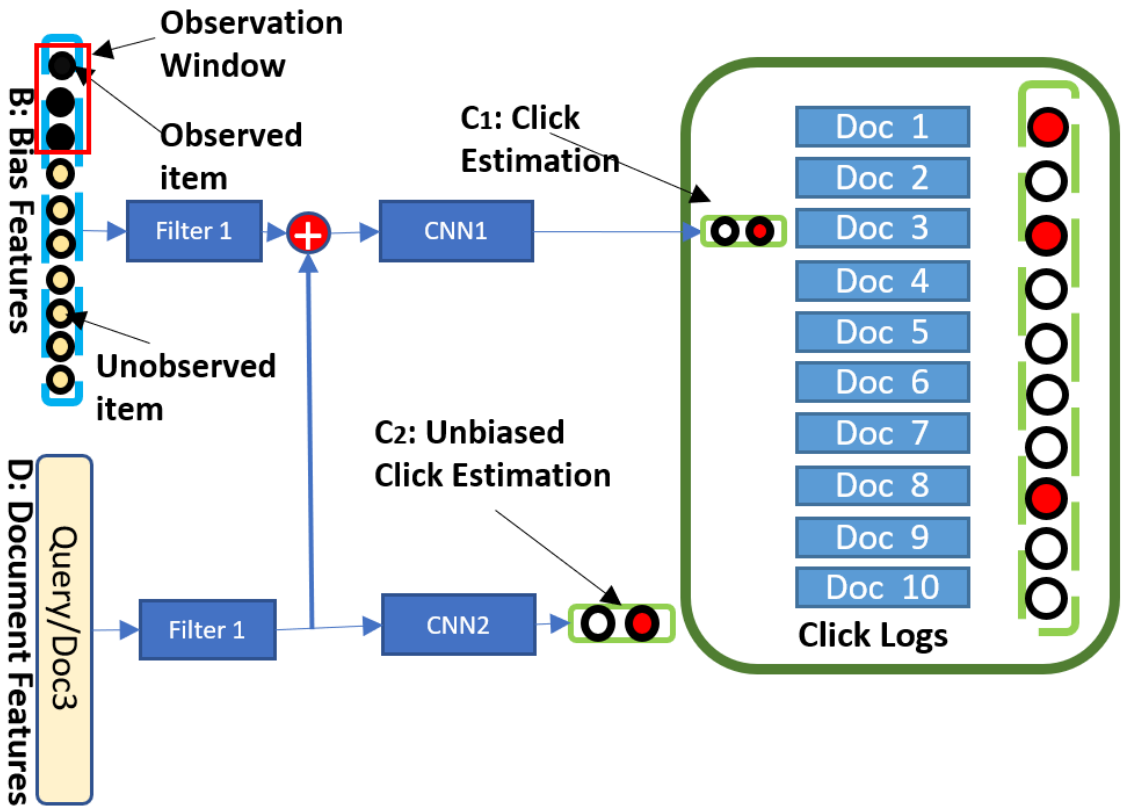

Figure 4.1: The structure of the DRLC value networks.

predictions from two value networks, we further update B. D is a vector, representing the features of a document. Those features are related to both the query and the document, such as the frequency of the query appearing in the document. The output of C_1 is the click estimation with bias or the possibility of clicking this document. The second CNN C_2 is a de-biased network, whose output is the de-biased click estimation or the possibility of clicking the item under the de-biased setting. The input of C_2 is just D. ‘ The structure of the value networks is summarized in Figure 4.1.

Pre-training: Since two networks focus on two aspects of the dataset, we pre-train them in different ways. We pre-train C_1 with the whole training dataset, which is a highly biased dataset. For the observation features B, we assume that the users observe the documents sequentially with an observation window. If the documents appeared in the window before, they are denoted as observed. If not, they are not

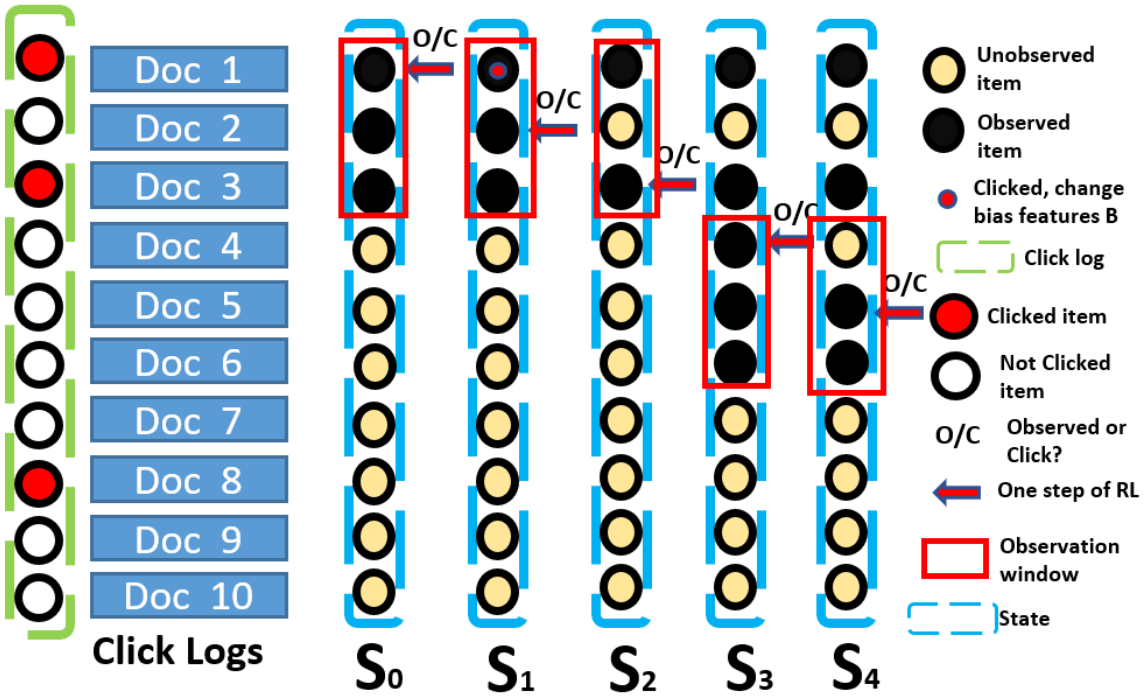

Figure 4.2: Illustration of the DRLC Reinforcement Learning process.

observed. Only when all the documents in the observation window are examined, the window moves to the next position, the process is presented in Figure 4.2. The size of the observation window is decided by the empirical estimation (In this research, the size of the window is 3.). C_2 is trained by a de-biased dataset. We assume the documents before the last clicked document is more likely to be observed. Therefore, we only pre-train C_2 with the documents before the last clicked document.

CNN architecture The input of the C_1 is an 100×1 vector (Bias features B) and a 56×1 vector (Document features D). The bias features are demonstrated in **Pre-train** part. The document features are generated from the URL provided in the dataset. The features we selected are identical to the ones listed in [106]. Firstly, we apply two filter layers to join the features from D and B. For each CNNs, they have three convolutional blocks. Each convolutional block contains 16 filters of kernel 3×1 with stride 1, a batch normalization layer, and a ReLU layer. The output layer is a fully connected network. The loss function is a softmax function. The structures of CNNs cannot be too deep, considering the scalability of the click model.

After pre-training, the networks are initialized to be trained by the RL method.

4.1.2 Reinforcement Learning (RL) Framework

The RL framework of DRLC is illustrated in Figure 4.2. The components of the RL are defined as follows:

State s is the click and observation state of the documents. We assume a user's attention can be modeled by an observation window, where users observe items in the observation window with a higher possibility.

Action a is selecting the state of a document at the position. Those states include: (Observed, Click), (Observed, Not click), and (Unobserved, Not click).

Transition T changes the document state in S based on a.

Reward R is how well the estimated click probabilities match the empirical distri-

bution observed in the click logs.

$$R_t = (C_t - C_{t,1})^2 + \beta O_t (C_t - C_{t,2})^2 \tag{4.1}$$

Where t is the position of the document. C_t is the click of the click logs. $C_{t,1}$ is the click prediction from C_1. $C_{t,2}$ is the click prediction from C_2. β weighs the importance of the de-biased prediction The value is set to 0.7 based on the validation part of the experiment. O_t shows that whether this document is observed. We have two assumptions of the observation. First, if the document is clicked in the click logs, it is observed. Second, if $C_{t,1}/C_{t,2} < \theta$, the document is unobserved. The first assumption is understandable. The second assumption is based on the bias effect, which is $P(O_t)C_{t,2} = C_{t,1}$. If $P(O_t)$ is small, it means the possibility of the observation is low. In this work, we empirically set θ as 0.3.

The goal of the RL is to learn a policy π^* to maximize $\mathbb{R} = \sum \gamma^t R_t$. In turn that means learning the *value* of each state, corresponding to click probability.

Update: The C_1 are further trained by the results of the final state S_T, where T is the total number of the documents in the click log. The C_2 is updated by the observed documents.

4.2 Experiments and Discussion

In this section, we introduce the datasets, metrics, and baselines.

Datasets: We demonstrate the effectiveness of the proposed model in two open-sourced dataset and one private dataset: ORCAS dataset [37], Yandex click dataset [117] and the real interactive dataset from a large e-commerce website (`https://www.homedepot.com/`). ORCAS is a click-based dataset associated with the TREC Deep Learning Track. It covers 1.4 million of the TREC DL documents, providing 18 million connections to 10 million distinct queries. The Yandex click dataset comes

Table 4.1: The search sessions information of the Home Depot website's Interactive data.

visitor ID	session id	date	time	searchterm
1000	1000-mobile-1	6/1/2020	6:30 pm	everbilt dropcloth
1000	1000-mobile-1	6/1/2020	6:34 pm	pull down shades
1001	1001-mobile-1	6/1/2020	6:36pm	fence panel
1001	1001-mobile-2	6/1/2020	6:36pm	fince dog ears

Table 4.2: The interaction information of the Home Depot website's Interactive data.

visitor ID	click sku	atc sku	order sku	product impression
1000	2034			3072—2034—2037—2036
1000	3022	3022	3022	3022—2051—3042—2071
1001				2030—1003—2029—1000
1001	2053			2055—2034—3034—2053

from the Yandex search engine, containing more than 30 million search sessions. Each session contains at least one search query together with 10 ranked items. The private Interactive Dataset (RID) is a 3-month search log. In this dataset, the users normally search for several queries. For each query, the search engine returns a list of products and then the user can interact with the results by clicking, adding the items to the cart and ordering. Table 4.1 and Table 4.2 shows a sample of the data. Additionally, with the products' ID, we can further find the page of the products and extract the text features. The features list can be found in [105].

Metrics We evaluate the model from two aspects. The first aspect is based on the click prediction. The second aspect is based on relevance. In terms of the click prediction, we use Log-likelihood and perplexity as the evaluation methodology. In terms of the relevance analysis, if the item is clicked, its relevance score is 1, or it is 0. Then, we consider the scores calculated to predict the click as the relevance score. We rank these scores and calculate the NDCG as our relevance prediction metric [13].

Baselines We use DBN, DCM, CCM, UBM and NCM as our baselines [23, 31, 41, 52, 13]. Those methods are the state-of-art click models based on PGM and neural networks. The parameter settings are based on the Pyclick package (`https:`

Table 4.3: The click simulation performance of DBN, DCM, CCM, UBM, NCM, URCM on ORCAS dataset, Yandex click dataset and the `https://www.homedepot.com/` e-commerce website interactive dataset (RID). The best performance results are highlighted in bold font. All improvements are significant with $p < 0.05$.

Dataset	Model	Perplexity	Log-likelihood
ORCAS Dataset	DBN	1.4628	-0.2273
	DCM	1.4647	-0.2894
	CCM	1.4664	-0.2778
	UBM	1.4593	-0.2203
	NCM	1.4545	-0.2186
	DRLC	**1.4326**	**-0.2037**
Yandex Click Dataset	DBN	1.3562	-0.2789
	DCM	1.3605	-0.3594
	CCM	1.3688	-0.3522
	UBM	1.3422	-0.2667
	NCM	1.3406	-0.2522
	DRLC	**1.3283**	**-0.2393**
RID dataset	DBN	1.3777	-0.2267
	DCM	1.3764	-0.2873
	CCM	1.3872	-0.2983
	UBM	1.3899	-0.2637
	NCM	1.3937	-0.2433
	DRLC	**1.3554**	**-0.2232**

`//github.com/markovi/PyClick`). (We did not use CACM as our baseline, because it is a session search click model, not a single search click model [26].)

Results and Discussion The results of the experiment are summarized in Table 4.3 and Table 4.4. The empirical results show that DRLC outperforms all baselines in terms of clicking prediction by 3.4% to 5.2%. Based on the T-test, this improvement is substantial. For the ranking prediction, DRLC outperforms the other baselines mostly.

Based on the empirical results, DRLC can predict the click, both in accuracy and relevance, better than the previous methods. We attribute the improvement of the click prediction to our assumptions of observation. In past, the unobserved data is hard to train, because it is almost impossible to manually label the data as observed documents or unobserved ones. However, in our RL framework, the users browse the

Table 4.4: The ranking performance of DBN, DCM, CCM, UBM, NCM, URCM on ORCAS dataset, Yandex click dataset and the `https://www.homedepot.com/` e-commerce website interactive dataset (RID). The best performance results are highlighted in bold font. All improvements are significant with $p < 0.05$.

Dataset	Model	NDCG@1	NDCG@3	NDCG@5	NDCG@10
ORCAS Dataset	DBN	0.596	0.606	0.623	0.655
	DCM	0.609	0.618	0.639	0.662
	CCM	0.615	**0.626**	0.637	0.671
	UBM	0.599	0.608	0.628	0.656
	NCM	**0.617**	0.625	0.639	0.677
	DRLC	0.610	0.624	**0.645**	**0.686**
Yandex Click Dataset	DBN	0.702	0.724	0.766	0.841
	DCM	0.729	0.744	0.775	0.845
	CCM	0.746	0.757	0.779	0.848
	UBM	0.729	0.739	0.769	0.841
	NCM	**0.756**	**0.763**	**0.788**	0.846
	DRLC	0.729	0.754	0.776	**0.848**
RID dataset	DBN	0.543	0.578	0.598	0.605
	DCM	0.566	0.587	0.603	0.611
	CCM	0.511	0.601	0.608	0.621
	UBM	0.538	0.612	0.618	0.632
	NCM	0.556	0.617	0.623	0.638
	DRLC	**0.616**	**0.624**	**0.645**	**0.648**

results sequentially. It is reasonable to assume that the documents are unobserved when the users are still observing the top documents. In this way, we can train the networks to classify whether the documents are observed. Besides, when the de-biased prediction and the biased prediction have a huge difference, it also indicates that the document is highly possible to be unobserved.

Additionally, in the RL framework, the initial State is updated during the training process to further improve the value networks.

Chapter 5

A Document-level Interactive Search System

Document-level interactive search, or dynamic search is a fundamental type of interactive search. In this chapter, I expand my previous work [154] that published in the Web Conference to present an efficient document-level interactive system that we contribute to this direction.

To support complex search tasks, where the initial information requirements are complex or may change during the search, a search engine must adapt the information delivery as the user's information requirements evolve. To support this dynamic ranking paradigm effectively, search result ranking must incorporate both the user feedback received, and the information displayed so far. To address this problem, we introduce a novel reinforcement learning-based approach, RLIrank. We first build an adapted reinforcement learning framework to integrate the key components of the dynamic search. Then, we implement a new Learning to Rank (LTR) model for each iteration of the dynamic search, using a recurrent Long Short Term Memory neural network (LSTM), which estimates the gain for each next result, learning from each previously ranked document. To incorporate the user's feedback, we develop a

word-embedding variation of the classic Rocchio Algorithm, to help guide the ranking towards the high-value documents. Those innovations enable RLIrank to outperform the previously reported methods from the TREC Dynamic Domain Tracks 2017 and exceed all the methods in 2016 TREC Dynamic Domain after multiple search iterations, advancing the state of the art for dynamic search.

5.1 The RLIrank Framework

In this section, we apply reinforcement learning to model the important steps of dynamic search. As Figure 2.2 illustrates, the search system is *Agent*. The ranking process is an *Action*. The feedback from the users is the *Reward*. The details of the framework are summarized as follows:

Agent: The *Agent* is the search system. The *Agent* ranks the documents based on the following policy:

$$\pi(a|S,\phi) = \begin{cases} \pi_1 & \text{if } p < \epsilon \\ \pi_2 & \text{if } p \geq \epsilon \end{cases} \tag{5.1}$$

Where p decides the policy. ϵ is the threshold of the greedy strategy. When the *Agent* chooses π_1, the *Action* returns a random document. If the *Agent* chooses π_2, then the *Agent* returns a document based on the probability calculated by Equation 5.2.

$$Pr(a|S,\phi) = \frac{V_\phi(f(S,a))}{\sum_{a_i \in A} V_\phi(f(S,a_i))} \tag{5.2}$$

Where ϕ represents all the parameters of the deep value network V_ϕ, which is introduced in Section 4.2. S is the current *State*, a is the current action. $f(S,a)$ is the transition function that updates *State* S based on the action a.

State: *State* S represents the current system's search context, which includes two parts:

$$S = [R, q] \tag{5.3}$$

Where R is a list of ranking documents and q is the query. Since *State* contains both R and q, two kinds of actions are required to update *State*.

Actions: The first action, which we note it as a_r, is the intermediate action. In Figure 5.1, to generate a list of ranking documents in a search iteration, we first rank each document. a_r represents this ranking process. Based on the Policy 5.1, with *Action* a_r, if we choose a document, we denote it as d_r.

The other action updates the query after each search iteration, which is noted as a_t. With the *Action* a_r, we denote the new query as:

$$q_t = T(q, F) \tag{5.4}$$

Where T is the feedback function to reformulate a new query based on the original query q and the user feedback F, which is further illustrated in Section 4.3.

Transition Function: *Transition Function* f defines how the search system is transited to a new *State*. For different actions, the transition function works in different ways. Basically, the transition function pairs the documents and the queries and update *State*. We define the transition function f as:

$$f(S, a_r) = \{(d_1, q), (d_2, q), (d_3, q), ..., (d_k, q), (d_r, q)\} \tag{5.5}$$

$$f(S, a_t) = \{(d_1, q_t), (d_2, q_t), (d_3, q_t), ..., (d_k, q_t)\} \tag{5.6}$$

In Equation 5.5, *State* is updated by appending the new document to the ranked list. In Equation 5.6, State is updated by replacing the query with the new formulated query. The result of $f(S, a)$ is an ordered set of the document and query pairs, which is the input for the ranker.

Rewards: In different steps of the dynamic search, the *Rewards* from the search system are varied. Although the most obvious *Reward* is the feedback from the user,

the relevance scores available in the training process should not be ignored. During the training process, the relevance scores between the query and the document are available. The feedback, which is the relevance scores among different subtopics of the query and the documents are only accessible after a list of ranked documents is given. However, such a *Reward* is evaluated by pairing the new query and the document. Therefore, after ranking a document, we evaluate the action based on the given relevance score in the training process. We evaluate the action with different metrics, such as NDCG, α-NDCG and et al. We denote the *Reward* R as:

$$R = V_*(f(S, a)) \tag{5.7}$$

To utilize intermediate *Rewards* to rank the next document better after, we minimize the following loss function after each action:

$$L(S, a|\phi) = \min_{\phi} \left(V_\phi(f(S, a)) - V_*(f(S, a))\right)^2 \tag{5.8}$$

Where ϕ is the parameters of the deep value network V_ϕ. This loss function evaluates the distance between the real metric and the estimated metric. The real metric requires the relevance scores between the query and the document, yet the estimated metric or deep value network does not require the scores, which is important for the testing process or the unsupervised situation.

5.2 RLIRank: Learning to Rank with Reinforcement Learning for Dynamic Search

The RLIrank implementation includes three parts. First, we introduce a stepwise learning framework, which is designed to train the value network V_ϕ. In Section 4.2, we describe the configuration of RLIrank deep value network V_ϕ. Finally, we introduce

our definition of feedback function T. This approach incorporate user feedback and the last search iteration query to formulate a new one, which derives from the classic Rocchio algorithm.

5.2.1 The Stepwise Learning Framework

In this section, we create a new approach – the stepwise learning framework to train the deep value network V_ϕ.

The details of this learning framework are illustrated in Figure 5.1. First, a document is chosen from the candidate list. When the document is chosen, it is also removed from the candidate list. The document choosing process depends on a score from V_ϕ and the ϵ-greedy strategy. The V_ϕ is a model with multiple entries, such as RNN, LSTM, and et al. The number of their activated entries is determined by the number of the chosen documents.

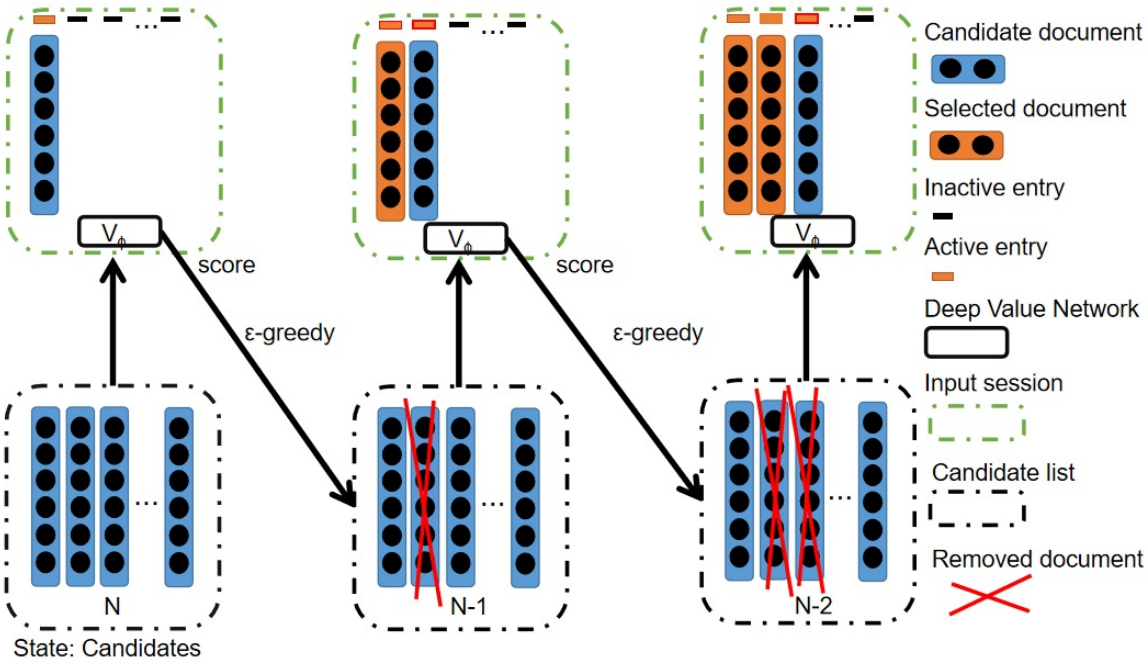

Figure 5.1: Illustration of the stepwise learning framework.

For instance, when selecting the first document, only one entry is activated. Thus

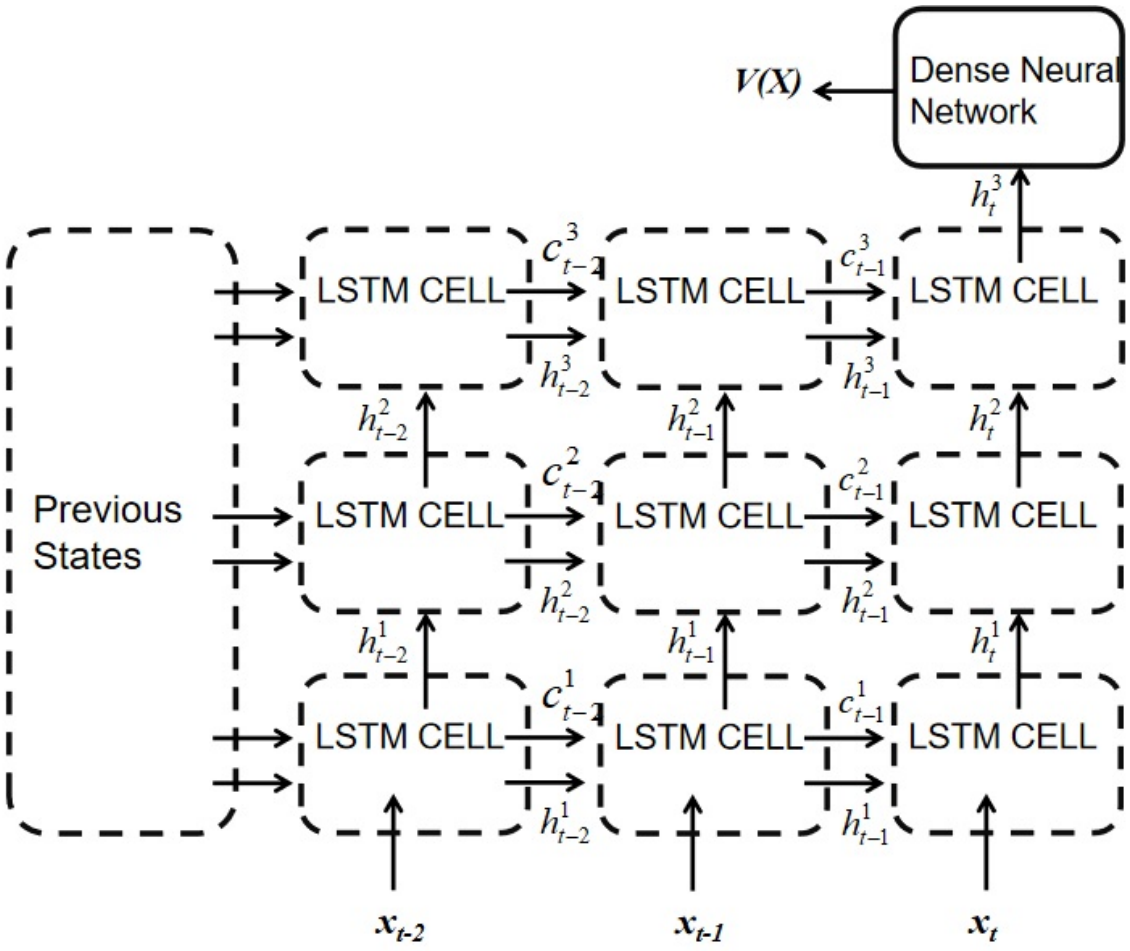

Figure 5.2: The structure of the stacked LSTM.

the input session is a list with one document. However, when we select the next input unit, the second entry is activated. The input session now is a list with two documents. Therefore, even though V_ϕ has many input entries, they are activated step by step.

Such a training approach can capture more information of the ranking than the traditional listwise method. For instance, 3 documents A, B, and C are given, whose relevance scores are 3,2,1. We consider the V_* is NDCG. Therefore, for this case, the learning goal is $3 + \frac{2}{\log_2(3)} + \frac{1}{\log_2(4)} = 4.7619$. However, if the relevance score of A is 4.2619 and B is 0. The list's score is still 4.7619. From this case, we can find that the full list of documents does not fully utilize all the information of every document

because based on a score of the list, it is impossible to recover every score for each document. However, with a pool of stepwise lists of documents, we can recover the scores for every document in any position.

5.2.2 Implementation of V_ϕ Network

In this work, we configure the V_ϕ with a stacked LSTM, which is introduced with three key parts: Input, Stacked LSTM Model Architecture and Loss Function.

Input: The input of the proposed stacked LSTM is dynamic because we apply a stepwise learning framework to train the stacked LSTM. We further justify the input as follows: Selecting the position: $X = X' \cup \{x_t\}$. X' is an order set of ranked items, x_t is the ranking item, $\cup$ means appending x_t to X'. From Figure 5.2, we find that the x_t is considered as the most recent input item of the LSTM. In the list of the ranked document, the nearer to the position t the item is, the more recent it is. The input of each cell x is an integration of a document and a query. For instance, if we use a vector d to represent the document, and use a vector q to represent the query, then $x = [d, q]$. Therefore, x is a query and document pair.

Stacked LSTM Model Architecture: The construction of this stacked LSTM is based on the demand of the experiment. The selection of the depth of the model is a trade-off between computation and performance. Based on the experiment and analysis in Section 6, we build a three-layer stacked LSTM as shown in Figure 5.2. The stacked LSTM contains some gate functions, allowing the cells to transfer the long term information or forget unnecessary information. The forget gate of jth layer, kth input j_k^j, the input gate i, the output gate o, the cell state c and the hidden state h of V_ϕ is computed as follows:

$$f_k^j = \sigma(W_f^j h_k^{j-1} + U_f^j h_{k-1}^j + b_f^j),$$

$$i_k^j = \sigma(W_i^j h_k^{j-1} + U_i^j h_{k-1}^j + b_i^j),$$

$$o_k^j = \sigma(W_o^j h_k^{j-1} + U_o^j h_{k-1}^j + b_o^j),$$

$$c_k^j = f_k^j \circ c_{k-1}^j + i_k^j \circ \tanh(W_c^j h_k^{j-1} + U_c^j h_{k-1}^j + b_c^j),$$

$$h_k^j = o_k^j \circ \tanh(c_k^j),$$

in which, $\sigma(x) = \frac{1}{1+e^{-x}}$ and applies to every entry. $h_k^0 = x_k$, the operator $\circ$ denotes the element wise product. The values of c_0^j and h_0^j are assigned by the Glorot Uniform Initializer in this work. The weight metrics and the bias vectors: W_f^j, W_i^j, W_o^j, W_c^j, b_f^j, b_i^j, b_o^j, and b_c^j are also assigned by the same initializer.

The output of the LSTM part is $LSTM(X', x_t) = h_K^J$, K is the ranking position, and J is the top layer's label.

Even with a list of documents, it is noteworthy that evaluating the current ranking document is still the most important, which means the effect of the ranked documents should be discounted by their positions. The earlier the document is ranked, the more the effect is discounted. Because we intentionally choose a one-directional model as our stacked LSTM, the information flow is sure to be discounted by the forget gates. **Loss Function:** In terms of the loss function of the stacked LSTM, it can be defined as:

$$L(S, a|\phi) = \min_{\phi} \left(V_\phi(X) - V_*(X)\right)^2 \tag{5.9}$$

V_ϕ is trained to minimize this loss function, in which V_* is a metric selected to evaluate the quality of the ranking results. We can define that $X = f(S, a)$.

5.2.3 An embedding Rocchio algorithm

We now introduce our new query reformulation method – the embedding Rocchio algorithm, which is the feedback function T in RLIrank.

To incorporate with V_ϕ, we need to embed the text contents to the numeric domain.

The embedding methods are determined by many aspects. For RLIrank, an important aspect to consider is the users' feedback.

In RLIrank, to digest the feedback from the users and make it observable for the *Agent*, we reformulate the query after each search iteration. The new query further optimizes the V_ϕ.

However, the query reformulation leads to an important issue. If we directly add words from relevant documents or remove them from the query, like the classic Rocchio algorithm, the variation is discrete, which easily results in unexpected results. Besides, the action of removing the words may be useless, because the query is so short that possibly does not contain the words.

To resolve the aforementioned problem, we further improve the classic Rocchio algorithm to an embedding Rocchio algorithm. This algorithm generates a new query based on the list of the ranked documents and the feedback. It is summarized as follows:

$$q_{n+1} = (1 - \gamma^n(b - c))q_n + \gamma^n(\frac{b\sum_{d_j \in D_r^n} d_j}{|D_r^n|} - \frac{c\sum_{d_k \in D_{nr}^n} d_k}{|D_{nr}^n|}) \tag{5.10}$$

Where q_n is the encoded query for the search iteration n. D_r^n is the documents receiving positive feedback from the users in the search iteration n. D_{nr}^n is the documents not receiving feedback or receiving negative feedback from the users in the search iteration n. d is the encoded documents. γ is the discount rate of different search iteration. b and c are the weights of the positive feedback and negative feedback respectively.

The query reformulation process of Equation 5.10 works in the following way. After a search iteration, we have search results, containing relevant documents and irrelevant documents. We calculate the mean of the relevant documents and irrelevant documents. Then, we weight them with b and c. With this operation, the query point in the encoded space moves close to the center of relevant documents by adding the mean of them and move away from the center of the irrelevant documents by

subtracting the mean of them. The moving process is discounted by γ, which weights the search iterations in the search session. It is noteworthy that the whole process is continuous. As a result, the embedding method of text content in RLIrank also requires to consider this continuity. Embedding space is normally discrete. However, when the embedding space is dense and large, it can be approximate to a continuous space. A possible option is using a universal sentence encoder.

5.3 Experimental Setting

In this section, we first introduce the basic experimental settings and datasets, then we experimentally compare RLIrank with different methods to report the main results.

The input of the stacked LSTM can be varied based on different tasks. For example, for the MQ2007&MQ2008, the input is a vector of 46 handcrafted features. For 2016&2017 TREC dynamic domain, it is an integration of the query's vector and document vector. The vector is encoded by an embedding method. For convenience, we denote them as input units. Each unit corresponds to a document.

5.3.1 Benchmark Datasets

There are two main datasets we consider: 2016 TREC Dynamic Domain (DD), 2017 TREC DD. They share similar datasets' settings. 2016 TREC DD contains Polar domain and Ebola domain. The Polar domain contains 26 queries and 1741530 documents. The Ebola domain contains 27 queries and 682157 documents. 2017 TREC DD dataset is the archives of the New York Times in 20 years, which has 60 queries and 1855658 documents [113]. Each query has several subtopics, and the documents have a relevant score for some subtopics provided by a *Jig* user simulator. The relevant score of a document to a query is the summation of all the scores of the subtopics. For the search engine, passages contents of the documents and the queries contents are

available. However, the relevance scores of the documents for each subtopic are given by a jig users simulator. The input of the simulator is a query's ID and 5 documents' ID. The output is whether your documents are relevant to the query and the scores of each document for the relevant subtopics ID. Therefore, during the search, the search engine does not know how many subtopics under the query and the contents of the subtopics. The scores of each subtopic can be used to calculate α-NDCG. Although the datasets have a similar setting, the 2016 DD track is α-DCG as the primary metric, and the 2017 uses $nSDCG$ [55][33].

RLIRank settings: The implementation of the proposed models is based on Tensorflow's basic models, and the parameters not mentioned are mostly the default settings. The stacked LSTM is constructed by 3 layers LSTM, 5 iterations, and a dense neural network. The dense neural network consists of 5 hidden layers and 1 softmax function. The neurons' numbers for LSTM cells or hidden layers are varied based on the experiments. The hidden neurons' number for each hidden layer is 1024, 512, 256, 16, 8, and the input size of the LSTM cell is 1024. The V_* of LSTM is α-DCG for 2016 TREC DD, and DCG for 2017 TREC DD. We apply the default Google Universal Sentence Encoder to transfer the words to the vectors. We encode both the query and document passage's content into a 512 vector and then integrate them into an input unit – a 1024 vector. The reason to use the universal encoder is that the value space of the encoded vectors is dense, which is suitable to apply the embedding Rocchio algorithm. Through little sample dataset test, we set γ of the embedding Rocchio algorithm is 0.9, b is 0.75, and c is 0.25. During the training process, $\epsilon = 0.5$. After each 1000 epochs, $\epsilon_{new} = \epsilon_{old} \times 0.9$. In testing process, $\epsilon = 0$. The drop-off rate of the neural network is 0.5, and all the activate functions in the neural network are ReLU. The model's training is stopped when the training improvement is less 0.01%.

5.3.2 Baselines

The baselines methods include the high-performing models appearing in the 2016 TREC Dynamic domain and 2017 TREC Dynamic Domain.

- **rmit-oracle-lm-1000:** The model firstly retrieves 1000 documents using Solr with the content language model. Then they use the ground truth to remove irrelevant documents from the initial list of documents. For each iteration, it returns the next 5 relevant documents from the initial list [7].
- **rmit-lm-nqe:** This method uses the Language modeling approach as implemented in Apache Solr using Dirichlet smoothing and default parameters. No query expansion (nqe) was applied [7].
- **ufmgHS2:** Hierarchical diversification with single-source subtopics and cumulative stopping condition [96].
- **ufmgHM3:** Hierarchical diversification with multi-source subtopics and window based stopping condition [96].
- **ictnet-emulti:** The model uses xQuAD and query expansion [150].
- **ictnet-params2-ns:** The model is the same as **ictnet emulti**, but changes parameters of other solutions. Not use stop strategy [150].
- **dqn-5-actions:** Use DQN to choose 5 possible search actions [83].
- **galago-baseline:** The first 50 results returned by galago [18].
- **RLIrank:** RLIrank Our method described above.

The five-fold validation method is applied to all the experiments. All baseline models are the most recent models or high-performance models. To confirm the significance of the improvements, we applied statistical significance testing (two-tailed Student's t-test) which reports p-values of < 0.05 for significant improvements.

Table 5.1: α-NDCG of baselines, and RLIrank on 2016 TREC Dynamic Domain dataset. The best performance results are highlighted in bold font. Results marked with * indicate significant improvements with $p < 0.05$.

	α-NDCG of 2016 TREC Dynamic Domain				
Iteration	RLIrank	rmit-oracle.lm.1000	ufmgHM3	ufmgHS2	rmit-lm-nqe
1	0.4947	**0.6874**	0.3516	0.3516	0.3581
2	0.6055	**0.7116**	0.4055	0.4079	0.3908
3	0.6500	**0.7251**	0.4306	0.4256	0.4073
4	0.7224	**0.7346**	0.4377	0.4335	0.4250
5	**0.7396**	0.7387	0.4417	0.4360	0.4322
6	**0.7445**	0.7406	0.4439	0.4366	0.4419
7	***0.7657**	0.7438	0.4446	0.4367	0.4461
8	***0.7685**	0.7448	0.4458	0.4367	0.4515
9	***0.7871**	0.7452	0.4468	0.4367	0.4555
10	***0.7927**	0.7468	0.4481	0.4368	0.4584

5.4 Results and discussion

Table 5.1 and Table 5.2 show that the improvement in the proposed RLIrank is substantial. Compared to the current state of art model, RLIrank improves up to 6.1% on the TREC 2016, and by over 20% on the TREC 2017.

We observe, on the TREC 2017 Dynamic domain, that RLIrank has a high first iteration performance. Since the performance of the other search iterations depends on the first iteration, the first iteration results are significant. The first iteration search is a pure LTR problem. Therefore, the improvement of the first search iteration results from the proposed deep value network.

We first explore the effect of the numbers of the stacked LSTM's layers. The results are presented in Figure 4. We find that as layers of the stacked LSTM increases, the performance improves. However, the improvement of the model decreases drastically when the number of layers is bigger than 3. To compromise the computation and the performance, we choose a 3-layer stacked LSTM as our deep value network. We further compare our proposed deep value network V_ϕ with the traditional listwise methods and an MDP method on the standard LTR benchmarks. This MDP method

Table 5.2: nSDCG of ictnet-params2-ns, ictne-emulti, galago-baseline, dqn-5-actions, clip-baseline, and RLIrank on 2017 TREC Dynamic Domain dataset. The best performance results are highlighted in bold font. Results marked with * indicate significant improvements with $p < 0.05$.

	nSDCG of 2017 TREC Dynamic Domain			
Iteration	RLIrank	ictnet-params2-ns	galago-baseline	dqn-5-actions
1	***0.6517**	0.4545	0.4337	0.4457
2	***0.6517**	0.4902	0.4590	0.4761
3	***0.6497**	0.4971	0.4594	0.4805
4	***0.6483**	0.4943	0.4526	0.4819
5	***0.6477**	0.4982	0.4506	0.4837
6	***0.6488**	0.4954	0.4519	0.4841
7	***0.6491**	0.4932	0.4548	0.4837
8	***0.6492**	0.4943	0.4564	0.4857
9	***0.6497**	0.5003	0.4592	0.4853
10	***0.6499**	0.5033	0.4581	0.4850

Table 5.3: NDCG of RankSVM, ListNet, AdaRank-NDCG, MDP, and RLIrank on MQ2007. The best performance results are highlighted in bold font. The results marked with * indicate significant improvements with $p < 0.05$.

	MQ2007				
Method	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5
RankSVM	0.410	0.407	0.406	0.408	0.414
ListNet	0.400	0.406	0.409	0.414	0.417
AdaRank	0.388	0.397	0.404	0.407	0.410
MDP	0.404	0.408	0.408	0.416	0.419
RLIrank	***0.434**	***0.440**	***0.444**	***0.446**	***0.451**

was the state of art of this benchmark. Table 5.3 and Table 5.4 shows that the improvements in RLIrank over other methods are substantial. Compared to the current reinforcement state of the art, MDP, RLIrank improves between 5.9% and 8.4% on the MQ2007 benchmark, and by 15%-19% on the MQ2008 benchmark.

The results prove that ranking model of RLIrank solves the traditional LTR problems well and thus promises the high performance of RLIrank in dynamic search problems. We attribute the improvements of it to the fact that while most of the baseline algorithms compared are list-based, the stacked LSTM is trained by more

Table 5.4: NDCG of RankSVM, ListNet, AdaRank-NDCG, MDP, and RLIrank on MQ2008 dataset. The best performance results are highlighted in bold font. The results marked with * indicate significant improvements with $p < 0.05$.

	MQ2008				
Method	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5
RankSVM	0.363	0.399	0.429	0.451	0.470
ListNet	0.375	0.411	0.432	0.457	0.475
AdaRank	0.383	0.421	0.442	0.465	0.482
MDP	0.409	0.435	0.463	0.481	0.510
RLIrank	***0.487**	***0.520**	***0.547**	***0.568**	***0.587**

completed information because of the stepwise training. Moreover, the structure of LSTM is also helpful for the deep value network to utilize the examined documents. Thus, a deep value network exploits the document feature similarity to promote or demote documents similar or diverse to the previously retrieved and rated documents.

To further justify the effect of the feedback method and the ranking model, we design 2 groups of experiments. We replace RLIrank's ranking model by MDP, which we call it RLIrank-MDP. Besides, we replace RLIrank's feedback organizer by the traditional Rocchio algorithm and a navie query expansion method. We denote them as RLIrank-Rocchio and RLIrank-nqe. We test them in the Ebola dataset's first 10 queries. The results are shown in Figure 5.3 and 5.4.

Figure 4(b) shows that RLIrank-MDP has a weaker performance, but improves faster, compared with RLIrank-Rocchio and RLIrank-nqe. At the last iteration, the performance of RLIrank-MDP is competitive. However, RLIrank outperforms all other variations. These results imply that the proposed deep value network and the embedding Rocchio algorithm can work separately, however, the combination of them can further boost each other.

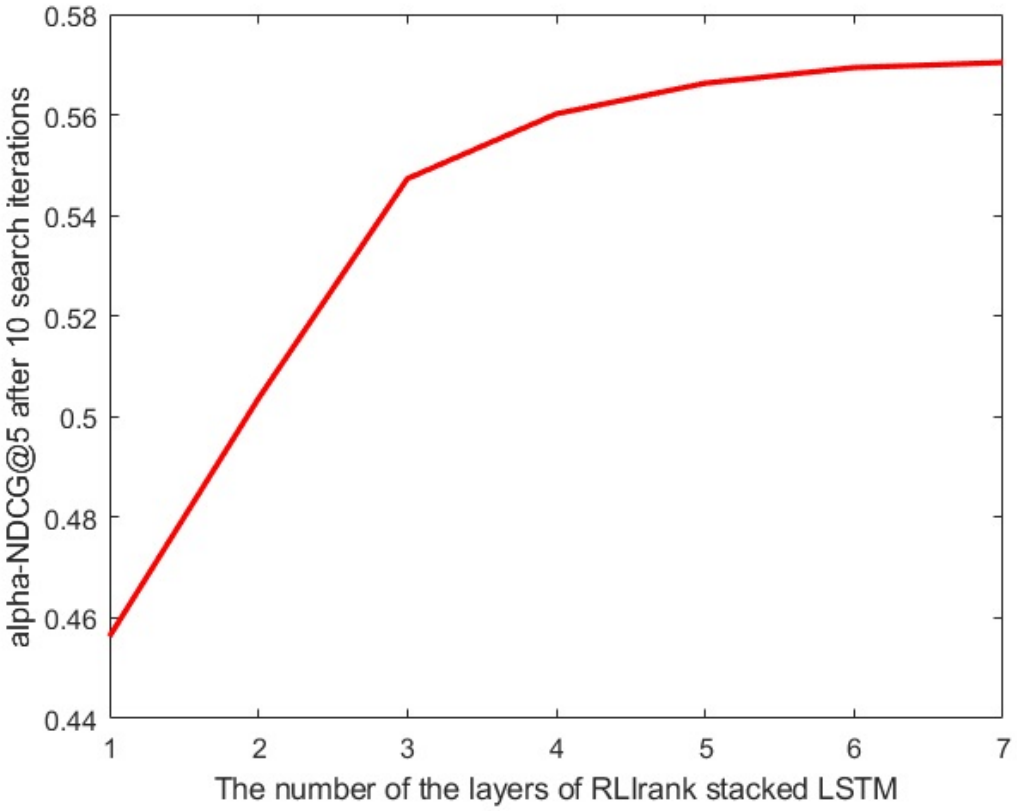

Figure 5.3: The improvement from the increment of the stacked layers of RLIrank's LSTM on Ebola 10.

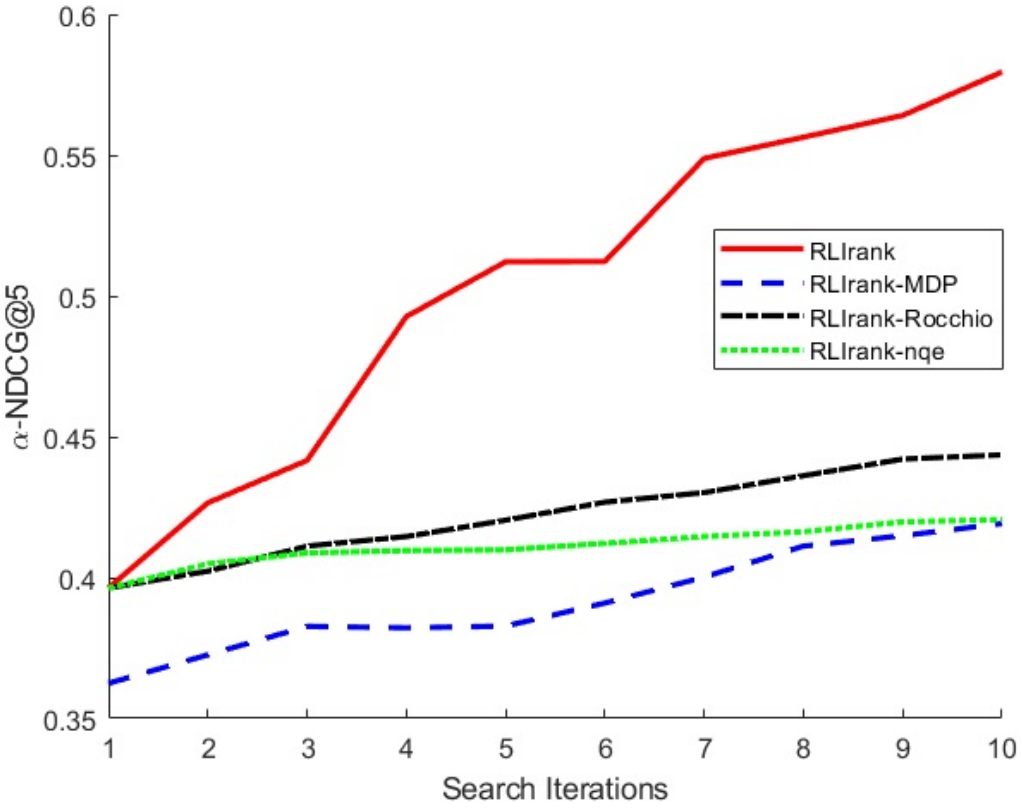

Figure 5.4: α-NDCG5 of RLIrank, RLIrank-MDP, RLIrank-Rocchio and RLIrank-nqe on Ebola 10.

Chapter 6

A Sentence-level Feedback Interactive Search System

Except document-level feedback interactive search system, sentence-level feedback is the other important type of interactive search. In this chapter, I will present a SOTA sentence-level interactive search system. This work is under review in the Web Conference 2023.

Interactive search can provide a better experience by incorporating interaction feedback from the users. This can significantly improve search accuracy as it helps avoid irrelevant information and captures the users' search intents. Existing state-of-the-art (SOTA) systems use reinforcement learning (RL) models to incorporate the interactions but focus on item-level feedback, ignoring the fine-grained information found in sentence-level feedback. Yet such feedback requires extensive RL action space exploration and large amounts of annotated data. This work addresses these challenges by proposing a new deep Q-learning (DQ) approach, DQrank. DQrank adapts BERT-based models, the SOTA in natural language processing, to select crucial sentences based on users' engagement and rank the items to obtain more satisfactory responses.We also propose two mechanisms to better explore optimal actions.

DQrank further utilizes the experience replay mechanism in DQ to store the feedback sentences to obtain a better initial ranking performance. We validate the effectiveness of DQRank on three search datasets. The results show that DQRank performs at least 12% better than the previous SOTA RL approaches. We also conduct detailed ablation studies. The ablation results demonstrate that each model component can efficiently extract and accumulate long-term engagement effects from the users' sentence-level feedback. This structure offers new technologies with promised performance to construct a search system with sentence-level interaction.

6.1 Methodology

This section proposes DQrank, a DQ model for sentence-based interactive search. In the interactive search setting, users search with a query and return sentence-level feedback like clicking or copying one or more sentences. In this section, we first introduce the DQ framework and how to apply it to interactive search. We then propose our sliding window ranking approach to explore the search space. Next, we present a method for identifying similar queries to avoid the cold-start problem. Finally, we introduce self-supervised learning and data augmentation techniques to improve the generalizability and robustness of DQrank.

The symbols used in this chapter are summarized in Table 6.1 and 6.2.

Symbol	Remarks
S; s_j	States; a state at iteration j
A; a_j	Actions; an action at iteration j
q	A query
f	A sentence in a document or feedback
f_i^j	The ith sentence in the feedback at iteration j
D_k; D	The kth ranking document; A document
x	The embedding result
X	The concatenation result
r; $R(a, q)$	Reward; The reward of action a given query q
d_k; d	The sentence representing document D_k; A sentence
$U(q, f)$	The user simulation function, given q and f
$u(q, D)$	The user simulation function, given q and D
$V(s, d)$	The discounted feedback function, given s and d
$Q(s, a)$	The Q value of s and a
$K(X)$	The weighted function of X
${}_i(s)$	The ith augmentation of s
ξ	The approximate factor of the reward function

Table 6.1: The important symbols of this chapter

Symbol	Remarks
ω	The neural networks weights of U
θ	The neural networks weights of Q
$L(\theta)$	A Loss function
π	A policy
γ	The Discount factor
N	Number of ranking results
E	Number of the feedback sentence
ψ	Similarity Threshold
c	Replay step
I	The initial ranking results (Candidate documents)
M	The number of sentences considered in a document
T_q	Candidate results for query q
Z	The replay memory
W_q	The existing ranking results in the dataset given q
P	The feedback pool

Table 6.2: The important symbols of this chapter.

6.1.1 Deep Q Learning Framework

In RL, intelligent agents take actions in an environment to maximize the notion of cumulative reward [71]. In particular, DQ is an RL method that uses a deep neural network to approximate the action gain Q [42]. Deep Q learning uses temporal difference (TD) learning to estimate the values Q with previous actions sampled by an experience replay approach. The objective of temporal difference learning is to minimize the distance between the TD-Target and $Q(s, a)$, suggesting a convergence of $Q(s, a)$ toward its actual values in the given environment.

In DQrank, we use DQ to model the interactive process for two reasons. First, DQrank has an experience replay mechanism, which is suitable for storing context information like feedback sentences and incorporating offline and online learning in interactive search. Second, the deep learning model in DQ can help better simulate the users' interaction, which can be used to expand the static search records. Therefore, DQ is appropriate for constructing our proposed model.

The DQ model consists of a set of states S, actions A, and a reward function R. In the context of the interactive search, the states S reflect the *query state*. This includes both query features and feedback sentences. We define the state s as:

$$s_t = (q, f_1^t, f_2^t, ..., f_E^t), \tag{6.1}$$

where s_t is the state at iteration t, q is the query and f_e^t is the eth feedback sentence at iteration t. E is the number of (stored) feedback sentences. The query part will remain unchanged throughout the process, but the feedback part can be updated. s

The action space A is the set of all possible ranking results. The initial retrieval will have some candidate items, I, and typically uses fast and straightforward methods to select these items from all possible items. Thus, the actions are the ordered subsets $A \subseteq I$ such that $|A| = N$, where N is the number of items DQrank presents to the users.

To select the action and compute the Q value, we design a user simulation function U for document ranking. The input of U is a query q and a sentence f from the document. $U(q, f)$ measure the relevance of q and f. While ranking, we calculate the U of the first M sentences of a document and select the highest score as the score for this document.

Finally, the reward $R(s, a, s')$ is the reward of a list of ranking results, measuring user engagement or other metrics related to the ranking quality. s' is the state before

action a and $a \in A$.

Our task is to find a policy π for the optimal ranking results. To achieve this, we organize the interactive search process as a DQ framework. We use a BERT-based model $Q(s, a; \theta_i)$ to estimate the Q function in Q learning. θ_i can be obtained from the experience replay.

The details of those parts are delineated in the following sections.

6.1.2 Training

We initialize a replay memory Z for the experience replay in RL training and the feedback pool P for the online serving system. All the transitions (s_{t+1}, a_t, r_t, s_t) are stored in Z and are randomly sampled as mini-batches to train the action-value function Q. The final states of each query are stored in the feedback pool P after every episode. In each episode, we process one query. The feedback pool P is used to accelerate and improve online learning.

The training of DQrank can be separated into two parts: (1) the offline training that can be done with a pre-trained users' simulation function, U, and (2) the online training to dynamically adjust to new search queries.

Offline Training. The goal of the offline training is to learn the policy π for the optimal ranking results based on the training dataset W_q. We first initialize the action-value function Q, the target action-value function $\hat{Q}$, and the state s_1. Any fast and straightforward method can be used to conduct the initial ranking to obtain the initial ranking results I_q for query q. In this work, we use BM25 [110]. However, since the size of I_q is still immense, we use U to re-rank and obtain a smaller candidate set T_q.

At the beginning of an iteration, the agent selects an action a based on the ϵ-greedy

policy π :

$$\pi = \begin{cases} \pi_1 & \text{with probability } \epsilon \\ \pi_2 & \text{with probability } 1-\epsilon \end{cases} \tag{6.2}$$

ϵ is a parameter to balance exploration and exploitation [108]. When the agent chooses π_1, we select a ranking result from W_q if W_q is not empty and then delete it from W_q. If W_q is empty, we generate a random ranking result by selecting items from T_q. If the agent chooses π_2, we generate a ranking result from T_q and try to maximize the Q value by re-ranking the results. In DQrank, we propose the sliding window ranking method see Section 6.1.6, an efficient approach to explore the RL action space.

After that, we can sample the transitions from Z and obtain their data augmentations for the experience replay. The data augmentations can improve the robustness of the model and will be demonstrated in Section 6.1.10. We minimize the difference between the TD-target and Q value (Equation 6.5) in the experience replay and update the target function $\hat{Q}$ every c step. After finishing T steps training, we store the best State of the query to the feedback pool P for the online serving system.

We then obtain the reward and store the transitions in Z. We later extract the samples from Z for the experience replay. The approach is summarized in Algorithm 1.

Online Training. Online training aims to obtain the state s for the new search queries. We first initiate the state s_0 with state retrieval, which can retrieve similar queries' states from the feedback pool P and will be discussed in Section 6.1.8. Then we go through the offline learning process without exploitation and model training. The feedback sentences f in state s are updated based on the users' feedback or interaction records. The transitions and final states are stored for the model refresh. Online training is more practical in industrious search tasks and much faster than offline training.

Algorithm 1 The Deep Q Learning Training Framework

Initialize the replay memory Z
Initialize the feedback pool P
Load users' simulation function U with pre-trained weights ω
Initialize action-value function Q with pre-trained weights θ (If the weights are not pre-trained, then generate them randomly)
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode=1:N **do**
    Initialize State $s_1 = (q, 0, 0, 0)$
    Initial ranking: Obtain candidate documents I_q with a weak ranker
    Re-ranking: Obtain candidate documents T_q from I_q with $U(q, D), D \in I_q$
    Loading the training dataset W_q
    for t=1:T **do**
        With policy π in Section 6.1.2 select an action a_t: Select a ranking result from W_q and delete it from W_q
        Or generate a random ranking result from T_q
        Or select an action a_t: Generate a ranking result from T_q to maximize $Q(s, a_t; \theta)$ based on the sliding window ranking introduced in Section 6.1.6
        Execute action a_t and obtain reward r_t and feedback sentences $f_1^{t+1}, f_2^{t+1}, f_3^{t+1}$
        Update the State $s_{t+1} = (q, f_1^{t+1}, f_2^{t+1}, f_3^{t+1})$
        Store transition (s_{t+1}, a_t, r_t, s_t) in Z
        Sample random mini-batch of transitions (s_{j+1}, a_j, r_j, sj) from Z, expand the samples with data augmentation in Section 6.1.10.
        Set $y_j = \begin{cases} r_j, & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-), & \text{otherwise} \end{cases}$
        Perform the gradient descent to $(y_i - Q(s_j, a_j; \theta))^2$ with respect to the weight θ.
        Update U with rearrangement learning in Section 6.1.7
        Let $\hat{Q} = Q$ every c steps
    end for
    Store s_{T+1} to P
end for

6.1.3 The Users' Simulation Model

The users' simulation function, U, mimics a user by determining whether the sentences in the document should be selected as the feedback sentence. In our model, U is a BERT-based classification function. Formally, the input of $U(q, f)$ is a pair of sentences. The first sentence q can be the query or the selected feedback sentence. The second sentence, f, is a sentence from a document D. Since the feedback sentences can reflect the users' satisfaction with the search results, the maximum probability scores of all the sentences in the document can be considered a metric to rank the documents. Therefore, the relevance between the q and a document D can be modeled as $u(q, D) = \max_{f \in D}(U(q, f))$.

For the input, q and f are separated by three tags, [CLS], [SEP], and [EOS], which represent the beginning, separator, and end of the sentence. The sentence embeddings are then encoded based on the tokens, segments, and positions as discussed in [133]. The embedding results from BERT, x, then represents the dependency between the f and d. We feed x to a fully connected neural network and calculate the selecting probability with the Softmax function, so $U = Softmax(W_T \times \tanh(x) + B_T)$. The U is further pre-trained by some question-answering and point-wise label datasets, which we introduce in Section 6.1.9.

6.1.4 The Q Function

The Q function learns the value of the partial action (Figure 6.1), a for a particular state, s. For ease, we denote f_0 as the query. The state is then $s = (f_0, f_1, f_2, ..., f_E)$. The action a is the list of ranked documents, $(D_1, D_2, ..., D_N)$. For document D_k, we select the sentence $d_k = \text{argmax}_{d \in D_k} V(s, d)$ where V is a discounted function based on the ranking metric as follows in Equation (6.3):

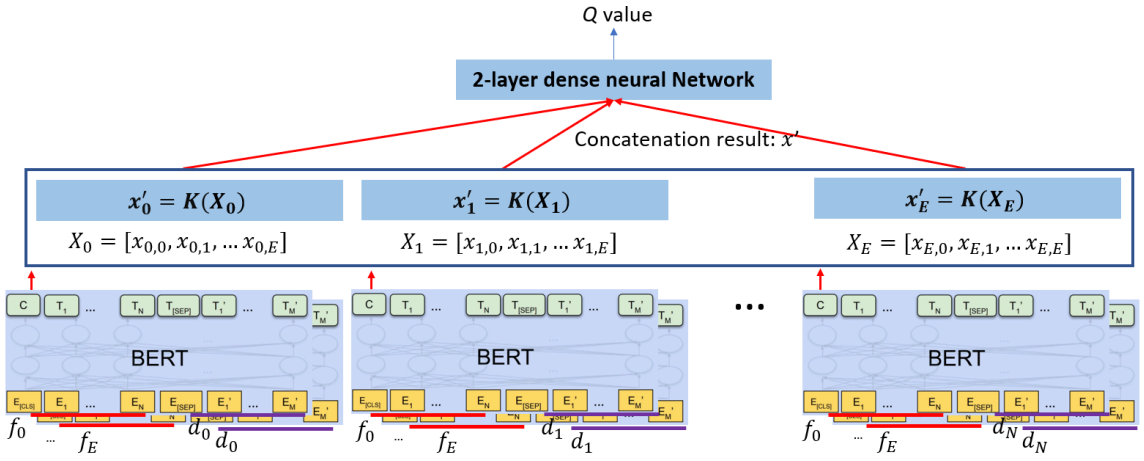

Figure 6.1: The Structure of Q function. f_0 is the query. f_e is the eth feedback sentence. E is the number of feedback sentences for the query f_0. d_k is the sentence representing the kth document in the ranking results, such that $d_k = \text{argmax}_{d \in D_k} V(s, d)$, $s = (f_0, f_1, ..., f_E)$. $k <= N$, N is the number of documents in the ranking results.

$$V(s_t, d) = \frac{1}{\sum_{e=1}^{E+1} \frac{1}{\ln(e+1)}} \sum_{e=1}^{E+1} [\frac{1}{\ln(e+1)} U(f_{e-1}^t, d)], \tag{6.3}$$

where d is a sentence of a document from the ranking result, d_i^j represents the jth sentence of the document ranking at position i in the searching results, and E is the number of feedback sentences.

Then we calculate the BERT embedding results,$x_{k,e}$, for every pair of f_e and d_k. Consequently, for every document k, we have $X_k = [x_{k,0}, x_{k,1}..., x_{k,E}]$. We further weigh those embedding results:

$$x'_k = K(X_k) = \frac{1}{\sum_{e=1}^{E+1} \frac{1}{\ln(e+1)}} \sum_{e=1}^{E+1} \frac{1}{\ln(e+1)} x_{k,e-1} \tag{6.4}$$

The weighted function K is based on Normalized Discounted Cumulative Gain (NDCG), a popular approach to measure the quality of a list of ranking results [64]. The proposed weighted function can re-balance the effect of the query and the feedback

sentences when E is of varying length. It also discounts the less important feedback sentences.

After the weighted embedding results are obtained, we concatenate them as the input to a 2-layer dense neural network used to estimate the Q value. The weights of the Q function can be obtained by minimizing the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\rho(Z)}[(y_i - Q(s,a;\theta_i))^2], \tag{6.5}$$

where samples come from the replay memory Z based on the random mini-batch sampling, ρ, and θ_i are all the neural weights at iteration i.

For every sample (s_{j+1}, a_j, r_j, s_j), if s_{j+1} is terminal, $y_j = r_j$, otherwise

$$y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-), \tag{6.6}$$

and weights of target action-value function $\hat{Q}$, θ^- is updated to θ every c steps. Since the BERT model is optimized multiple times in one step, we calculate the average values as the weights for the next step.

6.1.5 The Reward Function

The reward r evaluates the searching result a given the query q. Therefore, $r = R(s,a,s') = R(a,q)$. For some search results, we can find them in the dataset, W_q, and are already labeled. However, the agent generates many search results, and not all the labels exist in the dataset. To calculate the reward, we propose a reward transition method.

To estimate the reward $\hat{r}$ of an action a, first we extract a existed ranking result a'_i from the dataset W_q. We can calculate its DCG score with its query-document relevance scores. If the scores are not provided, we can use the user simulation function $U(q,D), D \in a'_i$. We assume the reward is a position-discounted score like

NDCG, then we have

$$\begin{aligned}
&\hat{R}(a,q) = \xi_i R(a'_i, q), \\
&\xi_i = \frac{\hat{R}(a,q)}{R(a'_i,q)} \approx \frac{\text{NDCG}(a,q)}{\text{NDCG}(a'_i,q)} = \frac{\text{DCG}(a,q)}{\text{DCG}(a'_i,q)} \\
&= \frac{\sum_{D_k \in a} \frac{u(q,D_k)}{\ln(k+1)}}{\sum_{D_k \in a'_i} \frac{u(q,D_k)}{\ln(k+1)}}
\end{aligned} \tag{6.7}$$

We can use the average estimation over W_q as our reward:

$$\hat{R}(a,q) = \mathbb{E}_{a'_i \sim W_q}(\xi_i R(a'_i, q)), \tag{6.8}$$

where W_q is the ranking dataset without deleting any element.

6.1.6 Sliding Window Ranking

Selecting an action to maximize the Q value is notoriously difficult for the ranking tasks [62]. To maximize the Q-value, we have to check every combination of the candidate documents. This colossal search space cannot be deployed to meet the latency requirements in most ranking systems. To effectively explore the searching space, we develop the sliding window approach by considering both users' simulation U and the Q functions. As introduced in Section 6.1.3, U can reflect the users' satisfaction with the item but cannot demonstrate the overall ranking performance. However, it is still helpful to generate approximate ranking results. Therefore, the first step is ranking the candidate documents with U.

Given the users' ranking, we design a sliding window of m, a parameter that reflects the users' attention. We start the window at the last item and move it backward, item by item, until we encounter the first item. Within each sliding window, we evaluate every combination to obtain the maximal Q value within the window. We replace the original order with this optimal order and proceed to the next window.

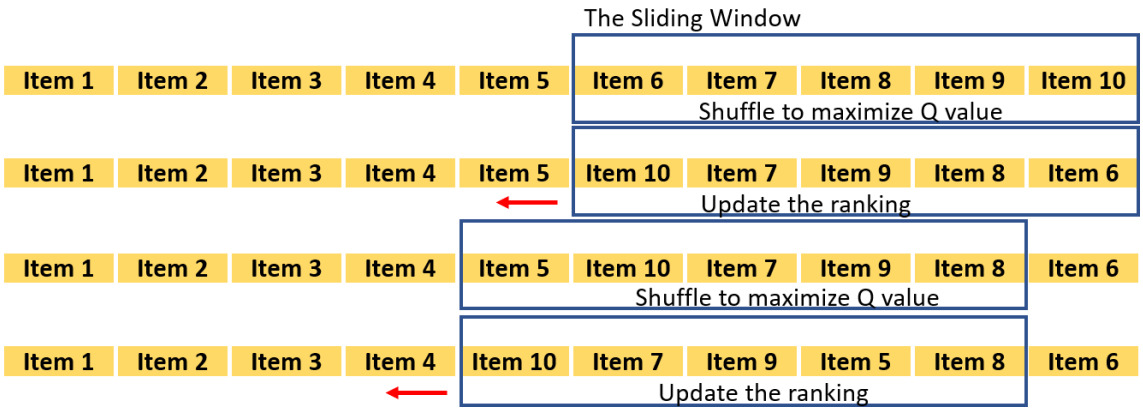

Figure 6.2: The Sliding Window Ranking Workflow

The workflow of the sliding window ranking is presented in Figure 6.2.

The advantages of this proposed method are three-fold:

1. This approach models the attention of the users. Users' attention is typically limited to a few items, so the position and observation bias is commonly generated within an attention window. Since we rank the items with the users' simulation function U first, it is reasonable to reflect the effect of ignored biases.

2. In most ranking tasks, the top-ranking items are usually the most important because users only browse the first serving page. This approach prioritizes the top-ranking items by ensuring items mistakenly ranked in the back can be moved forward.

3. Moreover, this approach is efficient enough to be deployed in the actual training process. If we assume that we have G items to rank and the number of our presenting items is N, we, in theory, would need to evaluate $C(G, m)$, where C denotes the number of combinations. However, under our approach, we only need to try it $(G - m + 1)m$ times, which is $\frac{1}{G-m+1}C(G, m)$ times faster. Since m is usually relatively small, the computation time of this ranking method can

be controlled in a reasonable scope.

To accelerate the computation, we calculate the embedding results and then apply this sliding window ranking method in the concatenation part to obtain different x'. By comparing the Q value, we can find the approximate optimal action.

6.1.7 Rearrangement Learning

Since users' feedback is mainly implicit, as the ranking results are usually passively digested [2], there can be discrepancies between the ranking using the users' simulation function, a_U, and the ranking results that maximize the Q value, because some relevant documents ignored by the users can be annotated as irrelevant. This inconsistency can introduce noise to the training and cause slow convergence and poor performance. Since we use U to estimate the optimal action and rank the documents, it is crucial to make a_U conform to a_Q. To achieve this, we train the function U by minimizing the loss function:

$$L(\omega) = \mathbb{E}_{d_{Q,j} \sim a_Q, d_{U,j} \sim a_U, f \sim S}[(U(f, d_{Q,j}; \omega) - y_{U,j})^2], \tag{6.9}$$

where $d_{Q,j}, d_{U,j}$ are the the sentences selected to represent documents $D_{Q,j}, D_{U,j}$ respectively; $D_{Q,j}, D_{U,j}$ are the jth documents in the ranking a_Q and a_U respectively; f is the query or the feedback sentences from state s, and $y_{U,j} = U(f, d_{U,j})$.

The rearrangement of U proposed in Equation (6.9) can be seen as a self-inference method to estimate the difference between simulated score U and the optimal score $\hat{U}$ that can rank the items to maximize the Q value. U is an auxiliary Q function to help us rank the items because directly using the Q function to rank the document is computationally impossible. Since U cannot incorporate some of the features from the other items, it cannot accurately estimate the Q value. However, rearrangement learning helps us model a probability distribution $\delta = \hat{U} - U$ that can close the gap

between U and $\hat{U}$. Noticeably, $\mathbb{E}(\delta) = 0$, because $\mathbb{E}_{d_{Q,j}\sim a_Q, d_{U,j}\sim a_U, f\sim S}[U(f, d_{Q,j}) - U(f, d_{U,j})] = 0$.

6.1.8 State Retrieval

The current framework assumes that users provide feedback throughout the entire reinforcement process, which in our experiments requires at least 15 iterations to converge. A user's search can be considered the first iteration of the RL with $\epsilon = 0$. In real-life search tasks, it is hard for the users to interact with the search engine for more than a handful of iterations. Thus, we propose to use state retrieval to identify similar queries to serve as the starting point for s_0.

Formally, let us denote the final state for query q as s_T. We store the BERT-encoded search query, $x(q)$, and s_T in our feedback pool P. When a new query is provided, q', we use BERT to encode the query, $x(q')$, and calculate the cosine similarity between this and all queries stored in P. If the highest cosine similarity is higher than our setting threshold ψ, we can retrieve this state as the initial state of the query. Consequently, the feedback sentences generated before can help improve the search.

6.1.9 Self-supervised Learning

Providing sentence feedback is essential in our users' simulation and relevance analysis. However, it is costly and challenging to collect sufficient annotations where sentences are labeled that help the search. Therefore, we use self-supervised learning to solve this issue.

Self-supervised learning is a method that can reduce the data labeling cost and leverage the unlabelled data pool [57]. In this work, we pre-train the users' simulation function U with the data of Question-answering (QA) systems. QA systems are sentence-based search systems. Only a few sentences in a document are considered

the critical answer to the question. In this work, we train the model by estimating the importance of the sentences in the document. Those sentences are labeled as 'Selected' or 'Not Selected.' After being pre-trained by the QA systems, the function U can be used to label the sentences crucial to the search, which are essential to training the DQ model further.

6.1.10 Data Augmentation

During the training process, DQrank is only exposed to a static set of queries (i.e., W_q). However, in practical settings, the search terms can be slightly different yet should still yield similar results. Therefore, to enhance the model's ability of generalization and robustness, we propose local exploration with data augmentations. This is done by reorganizing the search process of different sessions as an offline RL process to utilize the long-term benefit of the feedback sentences.

We design a new transformation $\hat{S}_t = (s_t)$, where $\hat{S}_t$ is the data augmentations of s_t, and the reward is smooth, that is $r(s_t) \approx r(\hat{s}_t)$. Note that the data augmentation cannot be too aggressive. Otherwise, it may hurt the performance of the model, because the reward for the original state does not coincide with the reward of the augmented state (i.e., $r(s_t, a_t) \neq r(\hat{s}_t, a_t), \hat{s}_t \in \hat{S}_t$). Therefore, the choice of needs to consider the connection between the reward and the state.

Since we cannot change the semantics of the original text and need to maintain the reward function remains smooth, we choose to paraphrase the sentences. "Paraphrase" expresses the meaning of a sentence using different words. We use the paraphrase technique to generate many queries with similar meanings to the query that goes through the RL framework. Those queries are used in the pre-training process of the users' simulation function U and the experience replay. We choose an advanced paraphrase toolbox Parrot as our paraphrasing model [39].

To reduce some noises and stabilize the training, we average the state-action values

and target values over different data augmentations of the state. We let

$$Q_t(s_t, a_t) = \frac{1}{|\hat{S}_t| + 1} \sum_{i=0}^{|\hat{S}_t|} Q({}_i(s_t), a_t)), \tag{6.10}$$

where $|\hat{S}_t|$ is the number of the data augmentations, ${}_0(s_t) = s_t$, and ${}_i(s_t), i \neq 0$ is the data augmentation of s_t. Since we assume the reward is locally smooth, r_t in Equation (6.5) remains the same.

6.2 The Experimental Setting and Results

In this section, we design experiments to demonstrate that DQrank can achieve better ranking performance in search tasks.

6.2.1 Datasets

We evaluate our model's ranking performance with three datasets. Those datasets evaluate the ranking results in different ways. The details are delineated as follows:
MS-MARCO: MS-MARCO datasets are large datasets designed for different information retrieval tasks [98]. We select the document retrieval dataset to evaluate the proposed model. This dataset contains 3.2 million documents and provides relevance scores for every pair of queries and documents. Therefore, we can calculate the overall performance of a ranking result with the normalized discounted cumulative gain (NDCG) score. At the same time, we use the passage retrieval dataset to fine-tune the users' simulation function U. The passage retrieval dataset has 8.8 million passages.
ORCAS: ORCAS is a click-log based dataset [37]. This dataset uses Indri [125] to retrieve 100 documents for every search. The clicks are used to evaluate these ranking results. The collection contains 20 million clicks, 1.4 million documents, and 10 million queries.

HITL[1]: The human-in-the-loop (HITL) dataset is an interactive search dataset from a CLIR (Cross-linguistics Information Retrieval) project. In this dataset, there are eight topics and eight queries per topic. For each query, volunteers are asked to select some relevant sentences from the top 10 documents as feedback. The number of documents is 30000, and 1053 sentences are selected as feedback sentences. The dataset belongs to a multi-university collaborative project in the USA and will be released soon.

We also use two additional datasets to fine-tune the users' simulation function U: ASNQ and ciQA. ASNQ is a high-quality natural language dataset for answer sentence selection, which consists of 59914 questions [45]. ciQA is a web-based dataset for training interactive question answering [74]. It contains 30 topics.

6.2.2 Experimental Settings

We use mean reciprocal rank (MRR) and NDCG of the top 10 items (nDCG@10) to measure the ranking performance. We use two point-wise ranking approaches and two state-of-the-art (SOTA) RL methods in search or recommendation systems as the baselines. Additionally, we evaluate the performance of three versions of our proposed model with different BERT models. The baselines include:

- **BM25** [110]: Baselines of the experiments .
- **BERT-U**: The proposed BERT-based user simulation model ranks the documents directly.
- **RLIrank** [154]: RLIrank is a policy gradient-based RL method, which is SOTA RL-based interactive search models.
- **SlateQ** [61]: SlateQ is a Q learning method with decomposition settings, which is SOTA RL-based recommendation systems.

[1] `anonymous.com`

- **DQrank (BERT)**: The DQrank model with U pre-trained using BERT [133].
- **DQRank (DistilBERT)**: The DQrank model with U pre-trained using DistilBERT, a smaller transformer version trained by distilling BERT [114].
- **DQrank**: The DQrank model with U pre-trained using RoBERTa, a BERT model trained with a larger dataset [89]. It is also our proposed implementation of DQrank.

The input embedding and structure settings are based on the [141].

We use 5-fold validation to examine our approach [70]. The data augmentations are only applied to the training part. The hyperparameters are determined by grid search [93]. We use the elbow method to determine the sliding window size, $m = 4$, from Figure 6.3. We use the Adam optimizer to train the model and the learning rate of 0.001. More details of other hyperparameters can be found in Table 6.3.

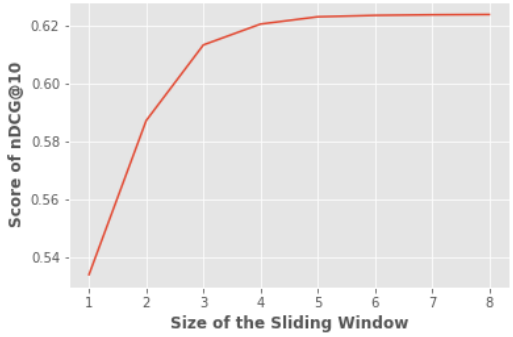

Figure 6.3: We sample 10% of the MS-MARCO dataset to explore the effect of the sliding window size on the ranking performance. Considering the trade-off between the ranking performance and the time complexity, we set the sliding window size as 4.

Hyperparameter	Value
Number of feedback sentences E	3
Two-layer Neural Network of Q	3072×56
Similarity Threshold ψ	0.85
Sliding window size	5
Replay step c	10
Size of action a	50
The maximum number of the sentences considered M	50
Experience replay batch size	32
The number of data augmentations	10
The policy selection probability ϵ	$0.05 + 0.85\mathrm{e}^{-\frac{T}{200}}$

Table 6.3: The Setting of Hyperparameters

6.2.3 Results

The ranking performance of the various models is presented in Table 6.4 and 6.5. We can observe that DQrank has more than 12% improvements over the previous SOTA RL approaches. Additionally, we found that the basic BERT-U model only has limited improvement from the benchmark BM25 (+2.6%), while the DQ framework brings more significant gains in ranking performance (+15.6% when comparing DQrank (BERT) with BERT-U). This observation indicates that the proposed learning framework and feedback sentence information are crucial for a better search system.

In terms of RLIrank and SlateQ, they outperform BM25 and BERT-U. However, they only have limited improvement (7.26%), demonstrating that the policy gradient approach and the DQ method without DQrank settings cannot utilize the interaction records sufficiently enough to drastically improve the interactive ranking performance.

Table 6.4: The comparisons on nDCG@10. The best results are in bold. Results marked with * indicate significant improvements with $p < 0.05$ than others.

Model	MS-MARCO	ORCAS	HITL
BM25	0.5246	0.5726	0.3424
BERT-U	0.5382	0.5897	0.3623
RLIrank	0.5627	0.6146	0.3829
SlateQ	0.5573	0.6068	0.3971
DQrank(BERT)	0.6221	0.6424	0.4422
DQrank(DistilBERT)	0.6154	0.6336	0.4402
DQrank	**0.6313***	**0.6517***	**0.4458***

Table 6.5: The comparisons on MRR. The best results are in bold. Results marked with * indicate significant improvements with $p < 0.05$ than others.

Model	MS-MARCO	ORCAS	HITL
BM25	0.2704	0.2844	0.2133
BERT-U	0.2871	0.2974	0.2245
RLIrank	0.3004	0.3116	0.2363
SlateQ	0.2987	0.3104	0.2411
DQrank(BERT)	0.3145	0.3287	0.2754
DQrank(DistilBERT)	0.3127	0.3265	0.2733
DQrank	**0.3224***	**0.3349***	**0.2773***

Both SlateQ and DQrank use deep Q learning to optimize the search, but SlateQ's decomposition method degrades the list-wise search results into point-wise ones. This prevents SlateQ from optimizing the search as a whole and can easily be affected by the search bias we introduced in Chapter 2.

Additionally, we compare the performance of the DQrank with different transformers. We can find that DQrank (our proposed implementation with RoBERTa) achieves the best performance. However, it only yields a 2.5% improvement over the other transformers. These results shows that a relatively lightweight BERT model, like distilBERT, can potentially replace RoBERTa without hurting the ranking performance. This can further accelerate DQrank and make it more practical and efficient in industrial search tasks.

6.2.4 Ablation Study

In this subsection, we investigate the effects of four essential components of DQrank: data augmentations (DA), state retrieval (SR), rearrangement learning (ARL), and self-supervised learning (SS). The models are listed as follows:

- **DQrank**: The RoBERTa-based DQrank model.
- **DQrank-SR**: The DQrank without the state retrieval mechanism (SR) introduced in Section 6.1.8.
- **DQrank-DA**: The DQrank without data augmentation training (DA) proposed in Section 6.1.10.
- **DQrank-DA-SR**: The DQrank without both SR and DA.
- **DQrank-SS**: The DQrank without self-supervised learning (SS) discussed in Section 6.1.9.
- **DQrank-SS-DA**: The DQrank without both SS and DA.

The results with and without rearrangement learning (see Section 6.1.7) are presented in Table 6.6. With rearrangement learning, DQrank-SR, which only uses data augmentations, provides an increase of 5.21% in nDCG@10 and 2.65% in MRR. When only using state retrieval (DQrank-DA), we increase 2.73% in nNDCG@10 and 1.32% in MRR.

We found that both state retrieval and data augmentation can improve search performance, and the latter is more helpful. However, when incorporating both approaches, we increase 9.58% in nDCG@10 and 4.64% in MRR, which is a significant improvement. This analysis reveals that data augmentations and state retrieval have substantial joint effects. The state retrieval can help us re-use those proper feedback sentences, and the local exploration from the data augmentations can connect those sentences to relevant documents.

Table 6.6: The ranking performance of DQrank with different components in the MS-MARCO dataset. ARL is rearrangement learning. All the results have significant improvements with $p < 0.05$ than DQrank-DA-SR without ARL.

Model	Without ARL	
	nDCG@10	MRR
DQrank-DA-SR	0.5406	0.2745
DQrank-DA	0.5554 (+2.73%)	0.2822(+2.80%)
DQrank-SR	0.5567 (+2.98%)	0.2831 (+3.13%)
DQrank	0.5597 (+3.53%)	0.2865 (+4.37%)
Model	With ARL	
	nDCG@10	MRR
DQrank-DA-SR	0.5677 (+5.01%)	0.2977 (+8.45%)
DQrank-DA	0.5832 (+7.88%)	0.3017 (+9.90%)
DQrank-SR	0.5973(+10.49%)	0.3056 (+11.32%)
DQrank	0.6221 (+15.08%)	0.3145 (+14.57%)

We also observe that all the scores decrease an average of 8% when we train the models without rearrangement learning. This observation shows that it is necessary to align the user simulation function ranking to maximize the Q value, as the gap can cause discrepancies to arise between the ranking results and yield suboptimal ranking results. Thus, rearrangement learning is significant for the users' simulation function U to support the documents' ranking.

Additionally, we explore the effect of state retrieval on the ranking performance during the online search sessions. As shown in Figure 6.4, in the initial search, both nDCG and MRR increase quickly from 0 to 5 iterations. However, DQrank has higher initial scores than DQrank-SR, which are +5.7% in nDCG@10 and +5.2% in MRR. After 30 search sessions, both models converge to relatively stable states, but DQrank is still around 1% better than DQrank-SR. This ablation study reveals that the feedback sentences collected from the offline search logs can help DQrank obtain better final states.

Lastly, we study the effect of self-supervised learning and data augmentations as a function of the number of search iterations t. The results are presented in Figure 6.5. We observe that data augmentations and self-supervised pre-training can improve the initial and final ranking performance by an average of 4%. At the same time, they have a combined effect of increasing both NDCG@10 and MRR by 6%. This ablation study shows that data augmentation and self-supervised learning can effectively boost the initial search performance while supporting the improved final search performance.

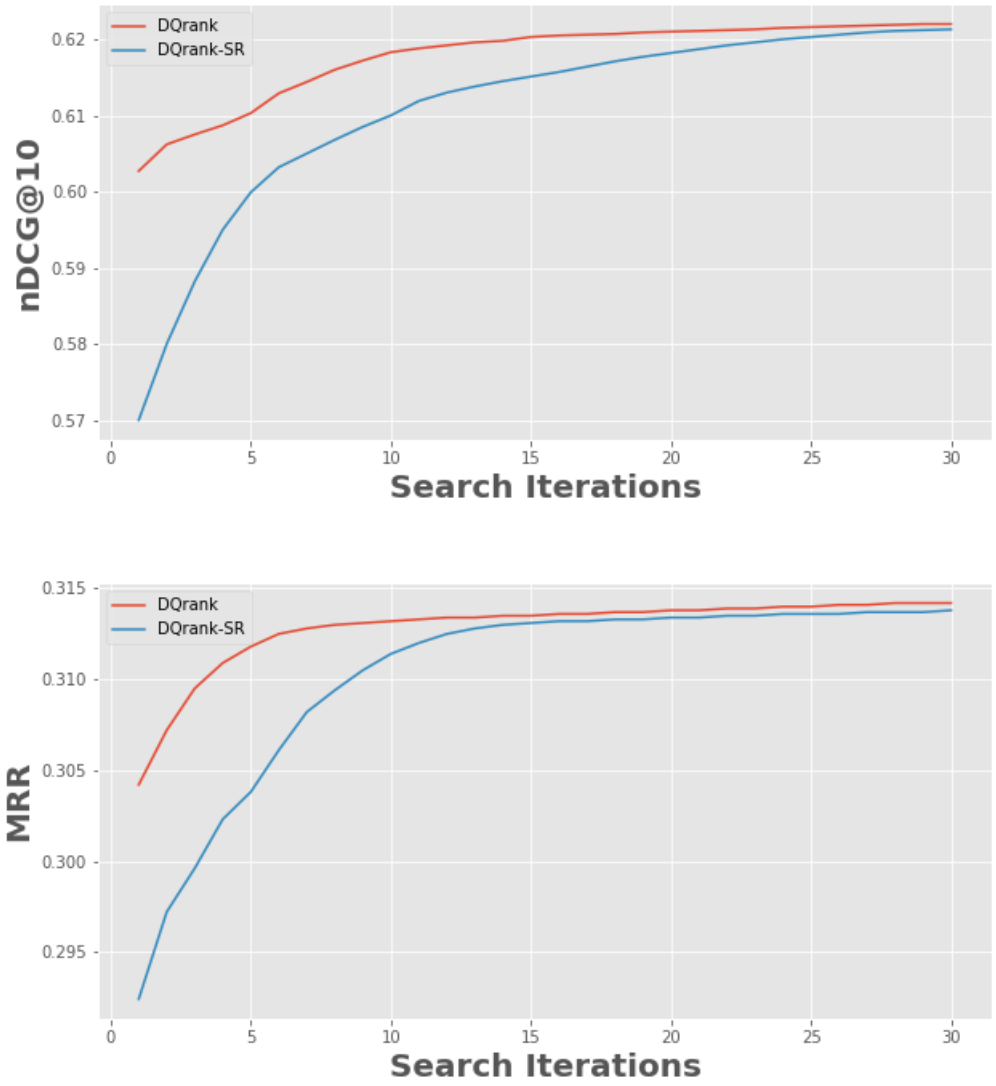

Figure 6.4: The ranking performance of DQrank with (red) or without state retrieval (blue) during the search session in the MS-MARCO dataset.

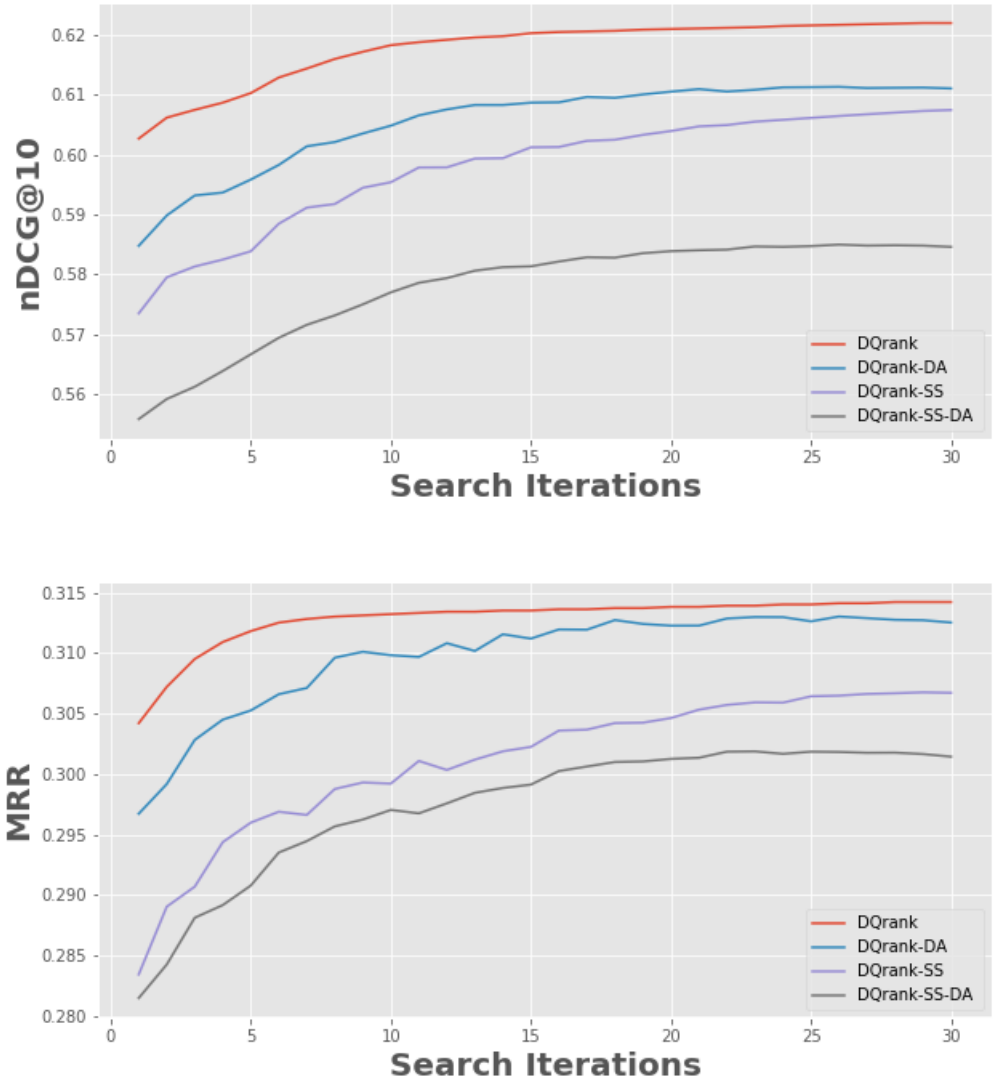

Figure 6.5: The ranking performance of DQrank with different components, DQrank (red), DQrank-DA (blue), DQrank-SS (purple), and DQrank-SS-DA (black), in the MS-MARCO dataset.

Chapter 7

Conclusion

This thesis presents my work on building efficient search systems with novel search diversification, search bias modeling, and interactive search approaches. The details of the contributions are listed as follows:

7.1 A new adapted mechanism to diversify search

In chapter 3, I adapt a mechanism for multi-aspect search results diversification from the Simpson's Diversity Index (SDI) in biology. This thesis uses theoretical analysis and experimental evaluation to demonstrate that the proposed method can diversify multi-aspect search faster and more effectively than some significant previous SOTA approaches. In contrast to the essential previous SOTA family of methods for diversification, the method optimizes both evenness and richness, which is the first time a search diversification approach considers these. The approach can significantly increase the amount of information in the search results, enabling users to provide more informative feedback in interactive search.

In this paper, we presented a novel approach to multi-aspect search results diversification, by adapting the Simpson's Diversity Index (SDI) from biology for this task. Unlike important previous SOTA family of methods for diversification, our

proposed method considers both evenness and richness. Based on theoretical analysis and experimental evaluation, we show that the evenness is significant for multi-aspect search, which is common in e-commercial product search. By defining the relevance and diversity in different ways, SDI can be further applied to other search and recommendation methods based on learning-to-rank or neural search and recommendation algorithms [158]. In the future, we plan to explore incorporating our SDI-based approach directly into learning-to-rank models, to further improve search and recommendation performance for e-commerce search and recommendation.

7.2 Search Bias Modeling

In this paper, we propose DRLC, which is a de-biased click model based on reinforcement learning. The model aggregates the advantages of the previous PGM methods and neural network methods, modeled by some novel assumptions of users' clicking and the observation bias. The empirical evaluation shows that DRLC is the state-of-art method in terms of click prediction, indicating that the RL, CNNs, and the proposed assumptions in this paper are helpful to improve the performance of the click models.

In the future, we can further consider other methods to de-bias the dataset. For example, we can apply counterfactual learning to train the de-bias network, which is theoretically unbiased [3]. DRLC can also be a trainer of some novel LTR models, providing a de-biased dataset.

7.2.1 An approach to reduce search bias

I delineate DRLC, a de-biased click model based on reinforcement learning in Chapter 4. The model incorporates the techniques of the previous PGM and neural network methods, modeling users' browsing patterns and detecting the observation bias. The

empirical evaluation shows that the approach can significantly reduce the observation bias and improve the ranking performance of the models using that de-biased feedback. Compared to the previous research, this technique can better reduce the feedback information bias, making this vital interactive search step more solid.

7.3 A document-level interactive search system

In Chapter 5, I described a new interactive search system, RLIrank. The approach is a reinforcement-learning-based ranking algorithm utilizing document-level feedback. It is specifically designed for the dynamic search where a user provides feedback for some documents after each search iteration. This approach can improve the search performance of the following search after receiving the document-level feedback of the last search more significantly than in the previous SOTA research.

Since interactive search is a relatively complex system, we further conduct the ablation study on RLIrank and analyze the key components, which reveals that the T function and S function are the keys to connecting the feedback and the ranking.

We proposed a new dynamic reinforcement-learning based ranking algorithm, RLIrank, specifically designed for dynamic search where a user provides feedback after each search iteration. We analyze the performance of the RLIrank reinforcement learning framework, and analyze the dynamic search setting, revealing the importance of T function and S function, showing the insights of the relationship between the feedback and the ranking. These two functions and the framework are helpful to direct future research in the dynamic search domain.

In this research, we only consider the performance of ranking, but some dynamic search tasks also consider the time to search or the length of the document, which should measure the model with other metrics, like Cube Test and Expected Utility.

In summary, our RLIrank algorithm provides a significant step towards dynamic

and contextualized interactive retrieval while opening up promising directions for future work. Additionally, RLIrank has two important contributions to improving the document-level interactive search:

7.3.1 A stacked LSTM trained by a stepwise learning framework

A stacked LSTM trained by a stepwise learning framework, which is the first attempt to incorporate the content (features) of the previously retrieved document into the ranking process. While the evaluation function ranks the documents and learns from the contents of the previously retrieved documents, the deep value network continuous to learn to continuously improve the ranking performance. We show that the deep value network significantly outperforms previous baselines, including a closely related previous state-of-the-art neural network-based method.

It is the first attempt to incorporate the content (features) of the previously retrieved document into the ranking process. In this framework, the deep value network continuously learns to improve the ranking performance while the evaluation function ranks the documents and learns from the contents of the previously retrieved documents. We demonstrate that the deep value network significantly outperforms previous baselines, including a closely related previous state-of-the-art neural network-based method.

7.3.2 An effective embedding Rocchio algorithm

This method allows RLIRank to outperform prior state-of-the-art methods for dynamic search significantly. Moreover, the experiments also imply that the embedding methods can significantly influence the performance of RLIrank. While the Rocchio algorithm is an efficient approach to reformulating the query, it is not appropriate to

extract document-level feedback information because it may also change the query and lead to instability. After our adaptation, we contribute a new Rocchio algorithm, which can generate more stable and continuous reformulated embedding queries.

An effective embedding Rocchio algorithm, which allows RLIRank to significantly outperform prior state of the art methods for dynamic search. Moreover, the experiments also imply that the embedding methods can significantly influence the performance of RLIrank. We apply a GCGAN to transform the document vectors to queries vector, which obtains striking improvement. In the GCGAN, we do not use labels information. However, during the training, the relevance scores between the documents and the queries are available. If this information is used in the GAN, it is possible to generate a better query transformer.

7.4 A sentence-level interactive search system

In Chapter 6, I introduced a new reinforcement learning approach for the search systems with sentence-level feedback, DQrank. This approach aggregates similar search sessions as an interactive process and optimizes the search results with deep Q learning by considering the feedback sentences.

This paper proposes a new reinforcement learning approach for the search systems with sentence-level feedback, DQrank. This method aggregates similar search sessions as an interactive process and optimizes the searching results with deep Q learning by considering the feedback sentences. Additionally, we provide three significant contributions to improving this RL framework: (1) We first introduce query reformulation as data augmentation to help the RL agent explore locally, making the search process stable and robust. (2) We propose sliding window ranking to estimate the optimal action efficiently. (3) We propose state retrieval to re-use the feedback sentences from the search history. In summary, our DQrank algorithm provides a cru-

cial step toward applying RL algorithms in search tasks while opening up promising directions for future work.

To resolve the notorious item selecting and queries robustness issues, we provide three significant contributions to improving this deep Q learning framework:

1. We introduce query reformulation as data augmentation to help the RL agent explore locally, making the search process stable and robust. Besides, this method helps us aggregate similar searches in an RL framework, which is crucial for reward estimation and actions selection.

2. We propose sliding window ranking to estimate the optimal action efficiently. This method can move the relevant items forward, securing the accuracy of the top-ranking documents while accelerating the ranking process.

3. We propose state retrieval to re-use the feedback sentences from the previous search. This method promises that the DQrank can serve users quickly and be practical in real-life search systems.

In summary, I proposes a new RL approach for the search systems with sentence-level feedback, DQrank. This method aggregates similar search sessions as an interactive process and optimizes the search results with DQ by considering the feedback sentences. Additionally, we provide three significant contributions to improving this RL framework: (1) We propose sliding window ranking to estimate the optimal action efficiently. (2) We introduce state retrieval to re-use the feedback sentences from the search history. (3) We reformulate the query using data augmentation to help the RL agent explore locally, making the search process stable and robust. Our experimental results demonstrate the performance of our model on three datasets. In summary, DQrank provides a crucial step toward applying RL algorithms in search tasks while opening up promising directions for future work.

7.5 Limitations

Although the proposed methods show strength in solving the key issues in the interactive search, they still have some limitations:

1. Simpson's diversity index method in Chapter 3 can only obtain the sub-optimal result of the binary quadratic problem. To archive better performance, the time complexity of the algorithm will increase drastically. A deep learning approach to explore the optimal may improve the sub-optimal with relatively lower effort.

2. DRLC approach in Chapter 4 focuses on solving observation bias. However, users' feedback may also be affected by position bias and other issues.

3. The experiments of RLIrank in Chapter 5 highly depend on the users' simulator from the dataset. However, most real-life search tasks do not contain such a simulator.

4. DQrank in Chapter 6 shows strengths in extracting interaction information better than the previous approaches. However, sentence-level feedback is still hard to obtain in most online search scenarios. Two solutions may overcome this limitation. First, joining more attention-detecting mechanisms into the search engines can help obtain sentence-level feedback efficiently, such as mouse tracking and eye tracking. Second, mixing with document-level feedback can help update the model when sentence-level feedback is absent.

7.6 Future Work

The work in this thesis demonstrates the tremendous potential of using users feedback to improve the search systems, which opens up promising direction for future work.

To extend my contributions, some work can be done in the future:

Diversifying the search results: In terms of the SDI model, the SDI score can be an efficient loss function in the search and recommendation methods based on learning-to-rank or neural search and recommendation algorithms [158]. To reach toward this goal, we will conduct more research on the binary quadratic optimization. We have two major directions:

- We can design a better representation of the diversity and relevance, so that the final loss function can be reorganized as SDI.
- We can deep learning model to fit the SDI function, which may achieve a better approximation.

De-bias the feedback: The de-bias network is not directly trained by some de-biased metrics, because those metrics are restricted by some incomplete assumptions [3]. However, we can consider a group of those approaches and use boosting methods to ensemble those algorithms and obtain a robust estimation of the bias.

The document-level interactive search: In this research, we only consider the performance of ranking, but some dynamic search tasks also consider the time to search or the length of the document, which should measure the model with other metrics, like Cube Test and Expected Utility.

The sentence-level interactive search summary, our RLIrank algorithm provides a significant step towards dynamic and contextualized interactive retrieval while opening up promising directions for future work.

All in all, this thesis contributes to the interactive search by proposing new methods to resolve the critical challenges in each component.

Bibliography

[1] Aman Agarwal, Ivan Zaitsev, and Thorsten Joachims. Counterfactual learning-to-rank for additive metrics and deep models. *arXiv preprint arXiv:1805.00065*, 2018.

[2] Aman Agarwal, Kenta Takatsu, Ivan Zaitsev, and Thorsten Joachims. A general framework for counterfactual learning-to-rank. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 5–14, 2019.

[3] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. Estimating position bias without intrusive interventions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 474–482, 2019.

[4] Sareh Aghaei, Kevin Angele, Elwin Huaman, Geni Bushati, Mathias Schiestl, and Anna Fensel. Interactive search on the web: The story so far. *Information*, 13(7):324, 2022.

[5] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W Bruce Croft. Unbiased learning to rank with unbiased propensity estimation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 385–394, 2018.

[6] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[7] Ameer Albahem, Damiano Spina, Lawrence Cavedon, and Falk Scholer. Rmit@ trec 2016 dynamic domain track: Exploiting passage representation for retrieval and relevance feedback. In *TREC*, 2016.

[8] Grigor Aslanyan, Aritra Mandal, Prathyusha Senthil Kumar, Amit Jaiswal, and Manojkumar Rangasamy Kannadasan. Personalized ranking in ecommerce search. In *Companion Proceedings of the Web Conference 2020*, pages 96–97, 2020.

[9] Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Kilian Weinberger. Learning to rank with (a lot of) word features. *Information retrieval*, 13(3):291–314, 2010.

[10] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[11] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3 (Feb):1137–1155, 2003.

[12] BioNinja. Brent cornell, 2022. URL `https://ib.bioninja.com.au/options/option-c-ecology-and-conser/c4-conservation-of-biodiver/biodiversity.html`.

[13] Alexey Borisov, Ilya Markov, Maarten De Rijke, and Pavel Serdyukov. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 531–541, 2016.

[14] Alexei Borodin. Determinantal point processes. *arXiv preprint arXiv:0911.1153*, 2009.

[15] 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.

[16] 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. ACM, 2007.

[17] David Carmel, Elad Haramaty, Arnon Lazerson, and Liane Lewin-Eytan. Multi-objective ranking optimization for product search using stochastic label aggregation. In *Proceedings of The Web Conference 2020*, pages 373–383, 2020.

[18] Marc-Allen Cartright, Samuel Huston, and Henry Feild. Galago: A modular distributed processing and retrieval system. In *SIGIR 2012 Workshop on Open Source Information Retrieval*, pages 25–31, 2012.

[19] Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: choice, discovery and relevance. 2011.

[20] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[21] B Chakraborty, R Kaustubha, A Hegde, A Pereira, W Done, R Kirlin, A Moghaddamjoo, A Georgakis, C Kotropoulos, and Pitas Xafopoulos. Bishop, cm, neural networks for pattern recognition, oxford university press, new york, 1995. carreira-perpiñán m., mode-finding for mixtures of gaussian distributions,

ieee transaction on pattern analysis and machine intelligence, vol. 22, no. 11, november 2000, 1318-1323. *IEEE transaction on Pattern Analysis and Machine Intelligence*, 22(11):1318–1323, 2000.

[22] Olivier Chapelle and S Sathiya Keerthi. Efficient algorithms for ranking with svms. *Information retrieval*, 13(3):201–215, 2010.

[23] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10, 2009.

[24] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 621–630, 2009.

[25] Olivier Chapelle, Thorsten Joachims, Filip Radlinski, and Yisong Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems (TOIS)*, 30(1):1–41, 2012.

[26] Jia Chen, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. A context-aware click model for web search. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 88–96, 2020.

[27] Laming Chen, Guoxin Zhang, and Hanning Zhou. Improving the diversity of top-n recommendation via determinantal point process. In *Large Scale Recommendation Systems Workshop*, 2017.

[28] Laming Chen, Guoxin Zhang, and Eric Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. In *Advances in Neural Information Processing Systems*, pages 5622–5633, 2018.

[29] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[30] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of systemics, cybernetics and informatics*, 8(1):43–48, 2010.

[31] Aleksandr Chuklin, Anne Schuth, Katja Hofmann, Pavel Serdyukov, and Maarten De Rijke. Evaluating aggregated search using interleaving. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 669–678, 2013.

[32] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. Click models for web search. *Synthesis lectures on information concepts, retrieval, and services*, 7(3): 1–115, 2015.

[33] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666, 2008.

[34] Kenneth L Clarkson. Coresets, sparse greedy approximation, and the frank-wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):1–30, 2010.

[35] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[36] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*, pages 87–94, 2008.

[37] Nick Craswell, Daniel Campos, Bhaskar Mitra, Emine Yilmaz, and Bodo Billerbeck. Orcas: 18 million clicked query-document pairs for analyzing search. *arXiv preprint arXiv:2006.05324*, 2020.

[38] Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

[39] Prithiviraj Damodaran. Parrot: Paraphrase generation for nlu., 2021.

[40] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung Yeung Shum. An empirical study on learning to rank of tweets. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 295–303, 2010.

[41] Georges E Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338, 2008.

[42] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[43] Zhichong Fang, Aman Agarwal, and Thorsten Joachims. Intervention harvesting for context-dependent examination-bias estimation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 825–834, 2019.

[44] Yue Feng, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. From greedy selection to exploratory decision-making: Diverse ranking with policy-value networks. 2018.

[45] Siddhant Garg, Thuy Vu, and Alessandro Moschitti. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. 2019.

[46] Gizem Gezici, Aldo Lipani, Yucel Saygin, and Emine Yilmaz. Evaluation metrics for measuring bias in search engine results. *Information Retrieval Journal*, 24(2):85–113, 2021.

[47] Dorota Glowacka, Tuukka Ruotsalo, Ksenia Konuyshkova, Samuel Kaski, Giulio Jacucci, et al. Directing exploratory search: Reinforcement learning from user interactions with keywords. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 117–128. ACM, 2013.

[48] Trey Grainger and Timothy Potter. *Solr in action*. Manning Publications Co., 2014.

[49] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[50] Dongyi Guan, Sicong Zhang, and Hui Yang. Utilizing query change for session search. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 453–462. ACM, 2013.

[51] Fan Guo, Chao Liu, Anitha Kannan, Tom Minka, Michael Taylor, Yi-Min Wang, and Christos Faloutsos. Click chain model in web search. In *Proceedings of the 18th international conference on World wide web*, pages 11–20, 2009.

[52] Fan Guo, Chao Liu, and Yi Min Wang. Efficient multiple-click models in web search. In *Proceedings of the second acm international conference on web search and data mining*, pages 124–131, 2009.

[53] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. A deep look into neural ranking models for information retrieval. *Information Processing & Management*, 57 (6):102067, 2020.

[54] Sonali Gupta. Ranking for better indexing in the hidden web. In *Dark Web Pattern Recognition and Crime Analysis Using Machine Intelligence*, pages 177–189. IGI Global, 2022.

[55] Matthias Hagen, Benno Stein, and Michael Völske. Webis at the trec 2010 sessions track. Technical report, BAUHAUS UNIV WEIMAR (GERMANY), 2010.

[56] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. *Advances in neural information processing systems*, 10, 1997.

[57] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. *Advances in neural information processing systems*, 32, 2019.

[58] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[59] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[60] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[61] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. 2019.

[62] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Morgane Lustman, Vince Gatto, Paul Covington, et al. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767*, 2019.

[63] Sina Jafarpour, Christopher JC Burges, and Alan Ritter. Filter, rank, and transfer the knowledge: Learning to chat. *Advances in Ranking*, 10:2329–9290, 2010.

[64] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[65] Gawesh Jawaheer, Peter Weller, and Patty Kostkova. Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 4 (2):1–26, 2014.

[66] Zhengbao Jiang, Ji-Rong Wen, Zhicheng Dou, Wayne Xin Zhao, Jian-Yun Nie, and Ming Yue. Learning to diversify search results via subtopic attention. In *Proceedings of the 40th international ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 545–554. ACM, 2017.

[67] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf

for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[68] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Acm Sigir Forum*, volume 51, pages 4–11. Acm New York, NY, USA, 2017.

[69] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789, 2017.

[70] Yoonsuh Jung. Multiple predicting k-fold cross-validation for model selection. *Journal of Nonparametric Statistics*, 30(1):197–215, 2018.

[71] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[72] Chris Kamphuis, Arjen P de Vries, Leonid Boytsov, and Jimmy Lin. Which bm25 do you mean? a large-scale reproducibility study of scoring variants. In *European Conference on Information Retrieval*, pages 28–34. Springer, 2020.

[73] Evangelos Kanoulas, Leif Azzopardi, and Grace Hui Yang. Overview of the clef dynamic search evaluation lab 2018. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 362–371. Springer, 2018.

[74] Diane Kelly and Jimmy Lin. Overview of the trec 2006 ciqa task. In *ACM SIGIR Forum*, volume 41, pages 107–116. ACM New York, NY, USA, 2007.

[75] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[76] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26 (2):195–197, 2008.

[77] Alex Kulesza. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2-3):123–286, 2012. doi: 10.1561/2200000044. URL `https://doi.org/10.1561%2F2200000044`.

[78] Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1193–1200, 2011.

[79] Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

[80] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015.

[81] Ching-Pei Lee and Chih-Jen Lin. Large-scale linear ranksvm. *Neural computation*, 26(4):781–817, 2014.

[82] 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.

[83] Yuanlong Li, Linsen Dong, Yonggang Wen, and Kyle Guan. Intelligent trainer for model-based reinforcement learning. *arXiv preprint arXiv:1805.09496*, 2018.

[84] HSUAN-TIEN LIN. Ordinal regression by extended binary classification. *Advances in neural information processing systems*, 2007.

[85] Feng Liu, Ruiming Tang, Xutao Li, Yunming Ye, Huifeng Guo, and Xiuqiang He. Novel approaches to accelerating the convergence rate of markov deci-

sion process for search result diversification. In *International Conference on Database Systems for Advanced Applications*, pages 184–200. Springer, 2018.

[86] Quan Liu, Xingcan Jia, Jiannong Quan, Jiayun Li, Xia Li, Yongxue Wu, Dan Chen, Zifa Wang, and Yangang Liu. New positive feedback mechanism between boundary layer meteorology and secondary aerosol formation during severe haze events. *Scientific reports*, 8(1):1–8, 2018.

[87] Rongjia Liu and Zhanxun Dong. A study of user experience in knowledge-based qa chatbot design. In *International Conference on Intelligent Human Systems Integration*, pages 589–593. Springer, 2019.

[88] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[89] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[90] Jiyun Luo, Sicong Zhang, and Hui Yang. Win-win search: Dual-agent stochastic game in session search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 587–596. ACM, 2014.

[91] Wing-Kin Ken Ma. Semidefinite relaxation of quadratic optimization problems and applications. *IEEE Signal Processing Magazine*, 1053(5888/10), 2010.

[92] Craig Macdonald, Rodrygo LT Santos, and Iadh Ounis. The whens and hows of learning to rank for web search. *Information Retrieval*, 16(5):584–628, 2013.

[93] Dhendra Marutho, Sunarna Hendra Handaka, Ekaprana Wijaya, et al. The determination of cluster number at k-mean using elbow method and purity evaluation on headline news. In *2018 international seminar on application for technology of information and communication*, pages 533–538. IEEE, 2018.

[94] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[95] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, 28 (1):1–16, 1982.

[96] Felipe Moraes, Rodrygo LT Santos, and Nivio Ziviani. Ufmg at the trec 2016 dynamic domain track. In *TREC*, 2016.

[97] Shashi Narayan, Shay B Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. *arXiv preprint arXiv:1802.08636*, 2018.

[98] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*, 2016.

[99] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[100] Harrie Oosterhuis, Rolf Jagerman, and Maarten de Rijke. Unbiased learning to rank: Counterfactual and online approaches. In *Companion Proceedings of the Web Conference 2020*, pages 299–300, 2020.

[101] Eli Pariser. *The filter bubble: How the new personalized web is changing what we read and how we think*. Penguin, 2011.

[102] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[103] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *ACM SIGIR Forum*, volume 51, pages 202–208. ACM New York, NY, USA, 2017.

[104] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*, 2019.

[105] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

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

[107] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

[108] Michael Rawson and Radu Balan. Convergence guarantees for deep epsilon greedy policy learning. *arXiv preprint arXiv:2112.03376*, 2021.

[109] Carlo Ricotta. Of beta diversity, variance, evenness, and dissimilarity. *Ecology and evolution*, 7(13):4835–4843, 2017.

[110] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[111] Tuukka Ruotsalo, Kumaripaba Athukorala, Dorota Głowacka, Ksenia Konyushkova, Antti Oulasvirta, Samuli Kaipiainen, Samuel Kaski, and Giulio Jacucci. Supporting exploratory search tasks with interactive user modeling. *Proceedings of the American Society for Information Science and Technology*, 50(1):1–10, 2013.

[112] Tuukka Ruotsalo, Jaakko Peltonen, Manuel JA Eugster, Dorota Głowacka, Patrik Floréen, Petri Myllymäki, Giulio Jacucci, and Samuel Kaski. Interactive intent modeling for exploratory search. *ACM Transactions on Information Systems (TOIS)*, 36(4):1–46, 2018.

[113] Evan Sandhaus. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752, 2008.

[114] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[115] Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. Classifying relations by ranking with convolutional neural networks. *arXiv preprint arXiv:1504.06580*, 2015.

[116] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[117] Pavel Serdyukov, Georges Dupret, and Nick Craswell. Wscd2013: workshop on web search click data 2013. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 787–788, 2013.

[118] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th inter-*

national ACM SIGIR conference on research and development in information retrieval, pages 373–382. ACM, 2015.

[119] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272, 2010.

[120] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[121] Edward H Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.

[122] *Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems*, 2002. Springer.

[123] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[124] Gray Stirling and Brian Wilsey. Empirical relationships between species richness, evenness, and proportional diversity. *The American Naturalist*, 158(3): 286–299, 2001.

[125] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the international conference on intelligent analysis*, volume 2, pages 2–6. Citeseer, 2005.

[126] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning

with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[127] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[128] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015.

[129] Zhiwen Tang and Grace Hui Yang. A reinforcement learning approach for dynamic search. In *Proceedings of the 26nd Text REtrieval Conference, TREC*, volume 17, 2017.

[130] Bart Thomee and Michael S Lew. Interactive search in image retrieval: a survey. *International Journal of Multimedia Information Retrieval*, 1(2):71–86, 2012.

[131] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to bm25 and language models examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, pages 58–65, 2014.

[132] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. Learning to rank by optimizing ndcg measure. *Advances in neural information processing systems*, 22, 2009.

[133] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[134] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity

with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.

[135] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. Position bias estimation for unbiased learning to rank in personal search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 610–618, 2018.

[136] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[137] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, volume 8, page 6, 2013.

[138] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992.

[139] Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. Reinforcement learning to rank with markov decision process. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 945–948. ACM, 2017.

[140] Sven Weinzierl, Matthias Stierle, Sandra Zilker, and Martin Matzner. A next click recommender system for web-based service analytics with context-aware lstms. 2020.

[141] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame,

Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[142] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, 2008.

[143] Pengtao Xie, Aarti Singh, and Eric P Xing. Uncorrelation and evenness: a new diversity-promoting regularizer. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3811–3820. JMLR. org, 2017.

[144] 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. ACM, 2007.

[145] Angela Yang and Grace Hui Yang. A contextual bandit approach to dynamic search. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 301–304. ACM, 2017.

[146] Grace Hui Yang and Ian Soboroff. Trec 2016 dynamic domain track overview. In *TREC*, 2016.

[147] Grace Hui Yang, Marc Sloan, and Jun Wang. Dynamic information retrieval modeling. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 8(3):1–144, 2016.

[148] Grace Hui Yang, Xuchu Dong, Jiyun Luo, and Sicong Zhang. Session search modeling by partially observable markov decision process. *Information Retrieval Journal*, 21(1):56–80, 2018.

[149] Tzu-Hsuan Yang, Tzu-Hsuan Tseng, and Chia-Ping Chen. Recurrent neural network-based language models with variation in net topology, language, and granularity. In *2016 International Conference on Asian Language Processing (IALP)*, pages 71–74. IEEE, 2016.

[150] Weimin Zhang, Yaokang Hu, Rongqian Jia, Xianfa Wang, Le Zhang, Yue Feng, Sihao Yu, Yuanhai Xue, Xiaoming Yu, Yue Liu, et al. Ictnet at trec 2017 dynamic domain track. In *TREC*, 2017.

[151] Yuchen Zhang, Dong Wang, Gang Wang, Weizhu Chen, Zhihua Zhang, Botao Hu, and Li Zhang. Learning click models via probit bayesian inference. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 439–448, 2010.

[152] Yuchen Zhang, Weizhu Chen, Dong Wang, and Qiang Yang. User-click modeling for understanding and predicting search-behavior. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1388–1396, 2011.

[153] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. " deep reinforcement learning for search, recommendation, and online advertising: a survey" by xiangyu zhao, long xia, jiliang tang, and dawei yin with martin vesely as coordinator. *ACM sigweb newsletter*, (Spring):1–15, 2019.

[154] Jianghong Zhou and Eugene Agichtein. Rlirank: Learning to rank with reinforcement learning for dynamic search. In *Proceedings of The Web Conference 2020*, pages 2842–2848, 2020.

[155] Jianghong Zhou, Jiangqun Ni, and Yuan Rao. Block-based convolutional neural network for image forgery detection. In *International Workshop on Digital Watermarking*, pages 65–76. Springer, 2017.

[156] Jianghong Zhou, Eugene Agichtein, and Surya Kallumadi. Diversifying multi-aspect search results using simpson's diversity index. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2345–2348, 2020.

[157] Jianghong Zhou, Sayyed M Zahiri, Simon Hughes, Khalifeh Al Jadda, Surya Kallumadi, and Eugene Agichtein. De-biased modeling of search click behavior with reinforcement learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1637–1641, 2021.

[158] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.