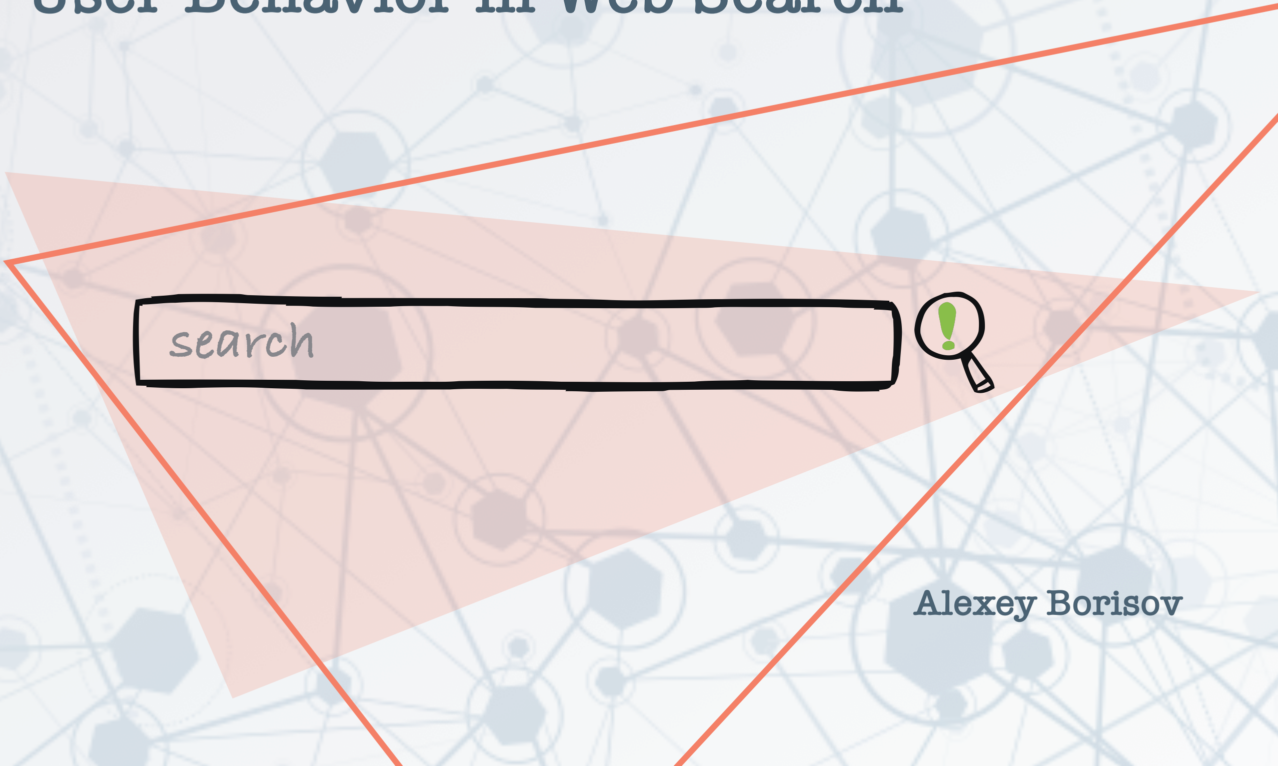
On Understanding, Modeling and Predicting User Behavior in Web Search



Alexey Borisov

On Understanding, Modeling and Predicting

User Behavior in Web Search

Alexey Borisov

On Understanding, Modeling and Predicting
User Behavior in Web Search

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op woensdag 3 oktober 2018, te 10:00 uur

door

Alexey Borisov

geboren te Kremenchuk, Oekraïne

The research was supported by Yandex.

Cover by Mostafa Dehghani and Samira Abnar
Printed by Off Page, Amsterdam, The Netherlands

ISBN: 978-94-6182-910-8

Acknowledgements

Good friends are like stars.
You don't always see them,
but you know they are always there...

Old Saying

First and foremost, I would like to express my deepest gratitude and appreciation to my promoter Maarten de Rijke. Dear Maarten, thank you so much for your patience, guidance, and support during all stages of my doctoral journey. I have always felt inspired after our meetings. Thanks for giving me the freedom in choosing research projects that I wanted. Thanks for teaching me to start every meeting with an agenda, to motivate experiments by research questions,[1] to encourage other people to work on an idea that I am passionate about, to split work and to collaborate on a paper. These skills have not only helped me in my research work but also contributed a lot to my personality. I am amazed by your ability to work for so many hours without getting tired. You have created a unique environment at the Information and Language Processing Systems (ILPS) group, which I had a great pleasure to be part of. I wish you a lot of success with the Innovation Center for Artificial Intelligence (ICAI), which, with you as a director, will soon become the most attractive place in the Netherlands to do world-class research in AI.

I am very much indebted to my co-promoter Pavel Serdyukov who organized the PhD program between Yandex and the University of Amsterdam. Dear Pavel, thank you for sending me long lists of scientific papers for every idea that we discussed at our weekly meetings. They helped me a lot to dive into new topics and to come up with interesting research questions.

I am very honored to have Thorsten Joachims, Evangelos Kanoulas, Mounia Lalmas, Christof Monz and Arnold Smeulders serving on my PhD committee. Thank you very much for reading and approving this thesis.

Dear Ilya, I would like to refer to you as my third scientific advisor, because your name appears in five out of six papers this thesis is based on. Thanks for sharing your interest in click modeling and for co-writing the most difficult parts of our papers over Skype. I am very grateful for your friendship and for the many fascinating and uplifting talks that motivated me to explore new things and discover new hobbies.

I would like to thank my other co-authors for their valuable contribution to the work described in the thesis and the work that contributed to the thesis indirectly: Bhaskar, Christophe, Julia, Martijn, Mostafa, and Tom.

Dear Tom, I have greatly enjoyed working with you. I like your approach to work, your attention to details, and your devotion to things that you love. You have had a difficult dilemma between your desire to work at Google and your love for Amsterdam. I am very impressed by the way you solved it. Working in the London office of Google during weekdays and spending weekends in Amsterdam is not going to be easy. But I

[1]This should be understood as "to motivate any practical work by the goals we want to achieve and to verify that the chosen plan is the shortest path towards achieving these goals."

have no doubt that you will make it work. Thanks for being my paranymph and helping me to organize the defense. Thanks for writing the *samenvatting* of this thesis and for being a part of my PhD journey.

Dear Julia, thanks for pushing me to finish the last two papers and for inviting me to co-organize the Search-Oriented Conversational AI (SCAI) workshop. You have an immense amount of energy! I wish you a lot of luck and success in the next chapter of your life. Keep being amazing, and thank you for the incredible discussions.

Dear Martijn, one of the papers this thesis is based on started as your MSc research project. I would like to thank you for the hard work you have done, for your enthusiasm, and for your excellent questions. This project helped me to develop new skills, and I am very grateful to you for choosing to work on an unpolished idea that Ilya and I came up with.

I thank all members of the ILPS group (ILPSers) and people around it for being part of my journey as a PhD student: Adith Swaminathan, Aleksandr Chuklin, Ana Lucic, Anna Seplyarskaya, Anne Schuth, Arianna Bisazza, Artem Grotov, Bob van de Velde, Boris Sharchilev, Chang Li, Chuan Wu, Christophe van Gysel, Daan Odijk, Damien Lefortier, Dan Li, Dat Tien Nguyen, David Graus, David van Dijk, Diederik Roijers, Dilek Onal, Evangelos Kanoulas, Evgeny Sherkhonov, Fei Cai, Hamid Ghader, Harrie Oosterhuis, Hendra Bunyamin, Hosein Azarbonyad, Ilya Markov, Isaac Sijaranamual, Ivan Zapreev, Jaap Kamps, Julia Kiseleva, Kaspar Beelen, Katya Garmash, Ke Tran, Lars Buitinck, Maarten de Rijke, Maarten Marx, Manos Tsagkias, Marlies van der Wees, Marzieh Fadaee, Masrour Zoghi, Mostafa Dehghani, Nikos Voskarides, Petra Best, Praveen Dakwale, Richard Berendsen, Ridho Reinanda, Rolf Jagerman, Shangsong Liang, Tobias Schnabel, Tom Kenter, Xinyi Li, Yaser Norouzzadeh, Yifan Chen, Zhaochun Ren and Ziming Li. Thank you so much for your friendship and for all those invaluable moments that shaped my personality and helped me to grow.

Dear Artem, I have always been impressed by your cheerfulness, your creative thinking, and your cooking skills. You are an amazing friend. Thanks for agreeing to be my paranymph and for all those fascinating and sometimes absurd talks that we had in the past four years. The PhD life without you would not be so bright!

Dear Mostafa, one of my first memories at ILPS is when I was sitting in front of you and Samira in the "fish tank" office. I love your scientific publications and your works as a graphical designer. I very much appreciate that you found time during your Apple internship to design the cover for this book. I want to thank you for your friendship and for being who you are: an easy, talented and integrated person.

I want to thank Richard for helping me to rediscover the beauty of chess and Xinyi for the stupendous cycling trips and the yummy dinners in Chinese restaurants.

I thank all members of the Yandex research team and research scientists in other parts of the company for helping me to grow as a researcher: Alexandr Vorobev, Alexey Drutsa, Andrei Mishchenko, Anton Dvorkovich, Anton Frolov, Artem Babenko, Boris Sharchilev, Damien Lefortier, Dmitry Ulyanov, Dmitry Vetrov, Elena Voita, Eugene Kharitonov, Gleb Gusev, Konstantin Lakhman, Ludmila Ostroumova, Michael Nokel, Pavel Kalinin, Pavel Serdyukov, Sergei Gubanov, Vadim Lebedev, and Valentina Fedorova.

Dear Artem, I am deeply grateful for your friendship and for the rare but very inspiring talks about science, literature, and life. I would also like to thank you for

introducing me to Pavel, and for being the best example of industrial researcher during my PhD years.

Dear Boris, I am very happy that you joined the same PhD program that I am finishing now. A lot of people have helped me during my PhD and I am very glad that I can pass on some of their advice to you. Although, I feel that so far I have learned from you more than I shared with you.

Dear Dmitry Vetrov, I very much appreciate the time when you were a part of our group. Thanks for the fruitful and enriching discussions about chatbots, graphical models, variational autoencoders and for your history lectures. I hope that we will get a chance to work together in the future.

Dear Anton Frolov, thank you for going with me through the most interesting and impactful papers of our time, for your critical feedback and for your invaluable career advice. I very much appreciate your friendship and I am looking forward to the technological miracles that we will make together at Yandex Zen!

I would also like to thank people who helped me make the decision to enroll in a PhD program: Dmitry Vatolin, Alexey Baytin, and Irina Galinskaya.

Dear Dmitry, thank you for supervising my first research project at the Moscow State University (MSU). I think that your group is a unique place at MSU where undergraduate students can do top-level research; and I am very happy that I have been part of your group.

Dear Alexey and Irina, thank you so much for giving me the opportunity to participate in an annual Machine Translation (MT) competition just a few months after I joined your department at Yandex. This allowed me to attend the largest international conference on Natural Language Processing (NLP), get inspired by the presentations there, and talk to many talented and motivated PhD students. I am deeply obliged to both of you for telling me after the conference “Lyosha, you need to do a PhD.” This was a turning point in my life.

I want to thank my friends for their continuous support and for listening to my PhD experiences. Special thanks go to my friends Sasha Diadishchev and Prohor Gladkikh. I would also like to thank Rachnah for helping me to write a motivation letter when I applied for the PhD position. This motivation letter turned out to be so poetic that I had to simplify it before sending to Pavel and Maarten.

I am very grateful for meeting Artem’s fiancée Yared and Ilya’s wife Maia, who became very good friends of mine. Dear Yared, thanks for inviting me to explore the Netherlands and showing me the best parts of the Valencia region. Thanks for your stories about Mexico and breathtaking pictures of Kamchatka (a beautiful region in the far east of Russia)! Dear Maia, thanks for the great walks and talks in Amsterdam and Saint Petersburg. I am looking forward to tangoing with you and Yared again.

I would like to say “muchas gracias” to Vanessa for the unforgettable moments that go beyond time and space...

Finally, I want to acknowledge my family. I thank my mother Alexandra for her unconditional love and support. Dear mom, thank you for always trying to find the best environment where I could learn and grow. Many people are surprised when I tell them how many things I have tried in my childhood. I thank my father Victor for giving me all his time when I was a kid and a teenager, for solving puzzles with me, for coming up with marvelous ideas that helped me to become who I am today. I thank my grandmothers Lilya and Anna for their love and prays and my grandfather Ruslan who passed away this year. I am sorry Ruslan, I wish I had visited you more often. I also thank my grandaunt Lisa for her everlasting optimism and happiness.

August 2018
Moscow *Alexey Borisov*

Contents

1 Introduction

Understanding people's continuously changing needs is one of the most challenging tasks in life [72]. Those who master this art are Titans, and they have the power to change the world. It does not matter whether you are an artist, a poet or a politician—the key to success lies in understanding the people around you. And those who fail to understand the needs of others, suffer from it every day. The current generation of artificial intelligence (AI) systems is still in its infancy in terms of understanding their users. Virtual assistants are able to handle a limited set of queries well, but they are not able to sustain long and meaningful conversations. Moreover, they are incredibly bad at predicting whether a particular recommendation (suggested at a given moment) will be appreciated by a user or will get the user frustrated. When virtual assistants are not able to interpret a user's utterance, they perform a web search and redirect the user to the search engine result page (SERP).

Web search engines provide quick and easy access to information available online. In the early days of the Internet, web links were the most valuable source of information for predicting result usefulness [127]. Nowadays, the whole web search stack relies on user behavioral data, starting from crawling policies to optimizing presentation of the results [1, 25, 69, 79–81, 93, 94, 101, 116, 126, 128, 161]. Furthermore, when a team of engineers proposes an improvement to web search algorithms, this improvement is tested using a so-called A/B experiment [69, 93, 94]. The users of the search engine are divided in two groups, A and B. The users in the first group (A) continue using the current version of the web search engine, while the users in the second group (B) start using the new version. After a certain period of time (typically a week), experts analyze the differences in user behavior in groups A and B, and based on this analysis make the final decision about whether the proposed improvement should be launched or not.

Since user logs are used both to improve and evaluate the quality of a web search engine, it is important to correctly interpret users' interactions with the system. A click on a search engine result indicates that a user is attracted by the result's snippet. And therefore the result is considered to be helpful to the user [27, 36, 46, 59, 60, 80, 104, 178]. However, if the user bounces back to the SERP and clicks on another result within a very short period of time (1 or 2 seconds), then the first click is likely to be unsuccessful [143].

In sponsored search, click-through rate (CTR) is one of the most important characteristics of advertisement quality [53, 64, 115, 135]. It is computed as the number of times an advertisement was clicked divided by the number of times the advertisement was served. In web search, however, comparing CTRs of different results (presented for the same query) is not straightforward due to various types of bias [30, 58, 81, 179]. For example, there is a *position bias*: users tend to click on the results presented on the top positions more often than on the results presented at lower positions [58, 81]. Other types of bias include *attention bias* towards visually salient results [30], *novelty bias* towards previously unseen results [179], etc. To account for these and other types of bias, click models have been proposed [34].

Click models are tools that allow us to extract unbiased user preferences from click logs [34]. Most existing click models make an *examination hypothesis* [36] that a user clicks on a result if, and only if, they examined the result's snippet and they are attracted by it. Under this assumption, click probability $P(C = 1)$ decomposes into the product of the probabilities $P(E = 1)$ that a user will examine the result's snippet and $P(A = 1)$ that a user will be attracted by it:

$$P(C = 1) = P(E = 1)P(A = 1). \tag{1.1}$$

Different click models use different strategies to model the examination probability $P(E = 1)$, while the result's attractiveness $P(A = 1)$ is usually modeled as a parameter $\alpha_{q,r}$ associated with the query-result pair, which does not depend on the position of the result. Parameters of click models are trained by maximizing the likelihood of the observed clicks. The examination component can be used to understand how a user's attention spreads over a SERP and how it changes during search. The inferred attractiveness parameters $\alpha_{q,r}$ are unbiased, which means that they can be used to compare results (w.r.t. the same query), without the need to account for position and other types of bias in the click logs [34].

In this thesis, we study existing tools for modeling and predicting user interactions with a search engine, improve them and develop new ways of gaining insights about user behavior. Our findings can potentially be generalized to other online applications, such as social networks and online advertising.

1.1 Research Outline and Questions

The main goal of this thesis is to create tools for understanding, modeling and predicting user interactions with a search engine. We begin with the observation that existing studies of click models assume that user click behavior does not change over time. This static assumption rarely holds in practice [133]. If the content of a search system changes (e.g., documents are added, altered or removed), user click behavior changes as well (users start clicking on new documents and stop clicking on the removed ones) [99]. Also, changes in a search algorithm (e.g., if documents are ranked differently) cause changes in user click behavior [69, 93]. Finally, document relevance is known to change over time, which in turn affects which search results users click on [43]. We study click models in an online scenario, where a search engine deals with a stream of click/skip observations. Specifically, we aim to find an answer to the following research question.

RQ 1 How to keep click models up to date with changes in search engine algorithms and user preferences?

We notice that most work dealing with click models lacks information about the choice of click model's hyperparameters. We anticipate that click models trained with suboptimal hyperparameters are likely to be badly calibrated. For example, a CTR model that predicts clicks with probability $\frac{\#\text{clicks}+\alpha}{\#\text{observations}+\beta}$ will be poorly calibrated if it is trained with wrongly chosen hyperparameters α, β and applied to examples with small numbers of observations. We investigate calibration properties of existing click models and ways to improve them. Specifically, we ask the following research question.

RQ 2 Does calibration help to improve click model performance and make it less dependent on the choice of hyperparameters?

Most click models are based on the probabilistic graphical model (PGM) framework, in which user behavior is represented as a sequence of observable and hidden events [34]. The PGM framework provides a mathematically solid way to reason about a set of events given some information about other events. But the structure of the dependencies between the events has to be set manually. Different click models use different hand-crafted sets of dependencies (represented by PGMs), while all of them are, by necessity, simplifications and likely to miss key aspects of user behavior. Neural networks, on the other hand, can discover the underlying patterns in the data by themselves. We introduce the idea of modeling user behavior as a sequence of distributed vector representations. And we seek to answer the following research question.

RQ 3 How to design a neural network that would be able to learn patterns in user click behavior directly from logged interaction data?

Click models focus on predicting an unordered set of clicks on SERP. Here, we make the first attempt to predict the order in which users will interact with search engine results. This new task is important, because it provides an opportunity for improving the user search experience. For example, knowing that there are high chances that a user will interact with the results in an order other than the one in which the results are presented on a SERP can be used by a search engine to proactively show an advice or make a change in the ranking. We formally define the task of predicting click sequences and design a neural network-based model to solve it. We aim to answer the following research question.

RQ 4 What are the challenges in predicting sequences of clicks and how to solve them?

Besides clicks, there are a lot of other behavioral signals that need to be understood and correctly interpreted. Here, we focus on times between user actions, which have been shown to provide a means to measure user satisfaction at the result level [48, 89], session level [48, 61] and system level [29, 141]. To interpret times elapsed between user actions, existing work uses mean values of the observed times [1–3, 26, 29, 61–63, 107, 108, 125, 141], or fits probability distributions to the observed times [89, 106]. This implies a *context-independence assumption* that the time elapsed between a pair of user actions does not depend on the context in which the first action takes place. We investigate whether this assumption really holds in practice and suggest a model that does not make this assumption. We ask the following research question.

RQ 5 How to correctly interpret times between user actions observed in different contexts?

Finally, we turn our attention towards content-based models or, as they are often referred to, latent semantic models. Existing content-based models can be categorized into two groups: unsupervised models (e.g., LSI [37], RLSI [160], PLSI [70], LDA [13, 162]) and supervised models using clicks or other user behavior signals for training (e.g., SSI [6], RMLS [167], BLTM [52], DSSM [75], CLSM [145]). Today, the use of latent semantic models by search engines is restricted to simply passing their outputs as features to a so-called global ranker, along with outputs of other models used for ranking. We argue that this is not optimal, because a single value output by a latent semantic model may be insufficient to describe all aspects of the latent semantic model's prediction. Specifically, we ask the following research question.

RQ 6 How to extract potentially useful information from a trained latent semantic model and how to utilize this information for improving ranking of search results?

1.2 Main Contributions

In this section, we summarize the main theoretical, algorithmic and empirical contributions of this thesis.

1.2.1 Theoretical contributions

1. We formulate the problem of keeping click models up to date with changes in user click behavior. We identify two main challenges: (i) how to efficiently incorporate newly observed information into a trained click model, and (ii) how to remove outdated information from a trained click model. We also propose an experimental protocol for evaluting click models in an online scenario, where a search engine deals with a stream of click/skip observations. Cf. Chapter 2.

2. We introduce the notion of calibration in the context of click modeling. And we advocate for making calibration a mandatory part of the click model evaluation protocol. Cf. Chapter 3.

3. We put forward the idea of using distributed representations to model user browsing behavior in web search. In contrast to existing click models, which represent user browsing behavior as a sequence of predefined binary events, the proposed representation is much richer, and thereby enables us to capture more complex patterns of user browsing behavior than existing click models. Cf. Chapter 4.

4. We formally define the problem of click sequence prediction and introduce the notion of probably correct click sequences. We also present a set of new prediction tasks to evaluate the quality of click sequence prediction. Cf. Chapter 5.

5. We introduce the notion of context bias in times elapsed between user actions, which has not previously been reported in the literature. And we describe four temporal prediction tasks to understand its implications. Cf. Chapter 6.

6. We argue that considering a single score of a latent semantic model (LSM) is not enough to determine its effectiveness in search, and that all potentially useful information captured by the model should be considered. We introduce the concept of *metafeatures*, i.e., feature vectors that describe the structure of the model's prediction, and present guidelines for creating them. Cf. Chapter 7.

1.2.2 Algorithmic contributions

7. We adapt online expectation-maximization (EM) techniques to efficiently incorporate new click/skip observations into a trained click model. We refer to the proposed algorithm as *Online EM*. Cf. Chapter 2.

8. To deal with outdated click information, we propose a modification to Online EM that discounts past observations depending on their age. We refer to the proposed algorithm as *EM with Forgetting*. Cf. Chapter 2.

9. We explain how to use *isotonic regression* to improve the prediction performance of existing click models. Cf. Chapter 3.

10. We introduce a *neural click model* (NCM) that uses a recurrent neural network (RNN) to explain user click behavior on a SERP. The proposed model allows us to directly understand user browsing behavior from click-through data, i.e., without the need for a pre-defined set of rules as is customary for PGM-based click models. Cf. Chapter 4.

11. We present a *click sequence model* (CSM) that predicts a probability distribution over click sequences. At the core of our model is a neural network with encoder-decoder architecture. We implement both the encoder and the decoder using RNNs. Cf. Chapter 5.

12. We propose a *context-aware time model* (CATM) that uses contextual information (and, in particular, previous user interactions with a search engine) for predicting and interpreting times between user actions. Cf. Chapter 6.

13. We instantiate the proposed *guidelines for creating metafeatures* using four latent semantic models. Cf. Chapter 7.

1.2.3 Empirical contributions

14. We show empirically that click models deteriorate over time if retraining is avoided. Cf. Chapter 2.

15. We evaluate the proposed *online EM* and *EM with Forgetting* using a publicly available click log of a major search engine. Our experimental results show that Online EM is orders of magnitude more efficient than retraining the model from scratch using standard EM, while losing little in quality. EM with Forgetting surpasses the performance of complete retraining while being as efficient as Online EM. Cf. Chapter 2.

16. We evaluate the proposed calibration method using a publicly available click log of a major search engine. Our experimental results show that (i) isotonic regression significantly improves click models trained with suboptimal hyperparameters; and that (ii) calibrated click models are less sensitive to the choice of hyperparameters than their original (non-calibrated) versions. Cf. Chapter 3.

17. We show empirically that the relative ranking of click models differs, depending on whether we use calibration or not. Cf. Chapter 3.

18. We compare the proposed neural click model (NCM) against traditional click models based on PGMs using a publicly available click log of a major search engine. Our experimental results show that NCM has better performance on the click prediction task (i.e., predicting user clicks on search engine results) and the relevance prediction task (i.e., ranking documents by their relevance to a query). Cf. Chapter 4.

19. We perform an analysis of the concepts learned by NCM, which shows that NCM learns similar concepts to those encoded in traditional PGM-based click models, and that it also learns other concepts that cannot be designed manually. Cf. Chapter 4.

20. We evaluate the proposed click sequence model (CSM) on a range of prediction tasks, namely predicting click sequences, predicting the number of clicks, predicting ordered/unordered sequences of clicks and, finally, predicting clicks themselves. Our experimental results show that CSM achieves state-of-the-art results on the unordered click prediction task used to evaluate click models and shows reasonable performance on the other tasks. Cf. Chapter 5.

21. We confirm empirically that there is a tangible context bias effect, which results in statistically significant differences in time-between-actions probability distributions for different contexts (in our case, for different sets of previous user interactions). Cf. Chapter 6.

22. We show that the proposed context-aware time model (CATM) provides a better means than existing methods to predict and interpret times between user actions. In particular, our experimental results show that CATM has better performance on the time prediction tasks (i.e., predicting time between user actions) and the relevance prediction task (i.e., ranking documents by their relevance to a query). Moreover, CATM allows us (i) to predict times between user actions in contexts, in which these actions were not observed, and (ii) to compute context-independent estimates of the times by predicting them in predefined contexts. Cf. Chapter 6.

23. We test the effectiveness of metafeatures by passing them to the global ranker along with the models' scores. Our experimental results show that through the use of metafeatures the performance of a combination of latent semantic models on the document ranking task can be improved as well as the performance of each individual latent semantic model by itself. Cf. Chapter 7.

1.3 Thesis Overview

We outline the research described in this thesis in Chapter 1. In Chapters 2 and 3, we discuss work that uses PGMs to explain user click behavior on a SERP. In Chapters 4, 5 and 6, we present work that uses neural networks (NNs) to interpret user behavior in web search. In Chapter 7, we describe work that uses gradient boosted regression trees (GBRT) to combine predictions of PGM- and NN-based latent semantic models. We draw conclusions and describe directions for future work in Chapter 8.

We recommend to read the research chapters in the order they are presented in the thesis. Figure 1.1 shows the dependencies between the research chapters. Chapters 2

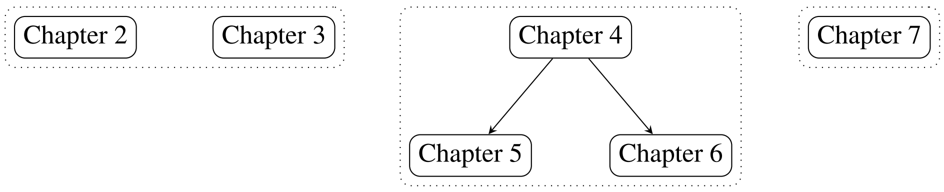


Figure 1.1: Dependencies between the research chapters.

and 3 can be read independently of each other. Chapters 5 and 6 should be read after Chapter 4.

1.4 Origins

In this section, we list publications that form the basis of this thesis. Each research chapter is based on a conference paper. We provide references to these publications and explain the roles of the co-authors.

Chapter 2 is based on I. Markov, A. Borisov, and M. de Rijke. Online expectation-maximization for click models. In *CIKM*, pages 2195–2198. ACM, 2017. Markov and Borisov wrote code and ran experiments. All authors contributed to the text.

Chapter 3 is based on A. Borisov, J. Kiseleva, I. Markov, and M. de Rijke. Calibration: A simple way to improve click models. In *CIKM*. ACM, 2018. Borisov and Kiseleva wrote code and ran experiments. All authors contributed to the text, Borisov did most of the writing.

Chapter 4 is based on A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A neural click model for web search. In *WWW*, pages 531–541. International World Wide Web Conferences Steering Committee, 2016. Borisov wrote code and ran experiments. All authors contributed to the text, Borisov did most of the writing.

Chapter 5 is based on A. Borisov, M. Wardenaar, I. Markov, and M. de Rijke. A click sequence model for web search. In *SIGIR*, pages 45–54. ACM, 2018. The main research question was formulated by Borisov and Markov. Borisov suggested the model architecture. Wardenaar wrote code and ran preliminary experiments. The main experiments were run by Borisov. All authors contributed to the text, Borisov did most of the writing.

Chapter 6 is based on A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A context-aware time model for web search. In *SIGIR*, pages 205–214. ACM, 2016. Borisov wrote code and ran experiments. All authors contributed to the text, Borisov did most of the writing.

Chapter 7 is based on A. Borisov, P. Serdyukov, and M. de Rijke. Using metafeatures to increase the effectiveness of latent semantic models in web search. In *WWW*, pages 1081–1091. International World Wide Web Conferences Steering Committee, 2016. Borisov wrote code and ran experiments. All authors contributed to the text, Borisov did most of the writing.

We also mention ten publications that contributed to the thesis indirectly.

- A. Borisov, J. Dlougach, and I. Galinskaya. Yandex school of data analysis machine translation systems for WMT13. In *WMT*, pages 99–103, 2013
- A. Borisov and I. Galinskaya. Yandex school of data analysis Russian-English machine translation system for WMT14. In *WMT*, pages 66–70, 2014
- M. Burtsev, A. Chuklin, J. Kiseleva, and A. Borisov. Search-oriented conversational AI (SCAI). In *ICTIR*, pages 333–334. ACM, 2017
- T. Kenter, A. Borisov, and M. de Rijke. Siamese CBOW: optimizing word embeddings for sentence representations. In *ACL*, pages 941–951, 2016
- T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *SIGIR*, pages 1403–1406. ACM, 2017
- T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *WSDM*, pages 779–780. ACM, 2018
- T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *ECIR*, page 837. Springer, 2018
- A. Voronov, A. Borisov, and D. Vatolin. System for automatic detection of distorted scenes in stereo video. In *VPQM*, 2012
- A. Voronov, D. Vatolin, D. Sumin, V. Napadovsky, and A. Borisov. Towards automatic stereo-video quality assessment and detection of color and sharpness mismatch. In *IC3D*. IEEE, 2012
- A. Voronov, D. Vatolin, D. Sumin, V. Napadovsky, and A. Borisov. Methodology of stereoscopic motion picture quality assessment. In *SPIE*. International Society for Optics and Photonics, 2013

2 Online Expectation-Maximization

Click models allow us to interpret user click behavior in search interactions and to remove various types of bias from user clicks. Existing studies of click models consider a static scenario where user click behavior does not change over time. In this case, click models are trained once using historical click data and are then used in various applications. In practice, however, user click behavior changes along with changes in searchable content, search algorithms, etc. For this reason, click models have to be constantly updated to correctly interpret changing user behavior. In this chapter, we answer the following research question asked in §1.1:

RQ 1 How to keep click models up to date with changes in search engine algorithms and user preferences?

We show empirically that click models deteriorate over time if retraining is avoided. We then adapt online expectation-maximization (EM) techniques to efficiently incorporate new click/skip observations into a trained click model. Our instantiation of *Online EM* for click models is orders of magnitude more efficient than retraining the model from scratch using standard EM, while losing little in quality. To deal with outdated click information, we propose a variant of online EM called *EM with Forgetting*, which surpasses the performance of complete retraining while being as efficient as Online EM.

2.1 Introduction

Click models have been developed to interpret user click behavior in search and to convert biased clicks on search results into unbiased estimates of the results' relevance [34]. Click models are used in various information retrieval tasks, such as ranking [27, 45], evaluation [32, 113] and user simulation [111, 169].

Existing studies of click models assume that user click behavior does not change over time. In this static scenario, click models are trained using a historical click log and evaluated on a limited test set of "future" clicks. This static assumption rarely holds in practice [133]. User click behavior changes over time and it is, therefore, important to keep click models up to date. This problem has been identified previously [104, 105], but has not yet been studied extensively. We recognize two challenges in keeping click

This chapter is based on Markov, Borisov, and de Rijke [114].

models up to date. First, how to efficiently incorporate newly observed information into a trained click model? Second, how to remove outdated information from a trained click model?

We consider the most widely used and effective inference technique for click models, i.e., expectation-maximization (EM), and study it in an online scenario. We propose two methods to adapt EM to online settings: Online EM, which efficiently incorporates new click information into click models, and EM with Forgetting, which deals with outdated clicks. We show experimentally that Online EM is orders of magnitude more efficient than complete retraining, while losing little in the quality of updated click models; EM with Forgetting can surpass the effectiveness of complete retraining.

2.2 Background and Related Work

Most click models operate with binary random variables and corresponding probabilities, e.g., how attractive a document is given a query, what the probability of examining a certain position on a SERP is, etc. These probabilities are the parameters of click models and they need to be estimated from observed clicks and skips.

The parameters of click models are usually calculated using either maximum likelihood estimation (MLE) or expectation-maximization (EM) [34]. MLE parameter estimation is used when a click model does not have hidden random variables. MLE does a single pass over a click log and calculates the model parameters directly based on the observed clicks. Thus, click models that use MLE are fast to train and straightforward to update in online settings. However, these models have been shown to perform worse than models that require EM for parameter estimation [57].

EM parameter estimation is used when a click model has hidden variables. EM estimates the model parameters iteratively. At each iteration, the current values of the parameters are fixed and new values are calculated based on user clicks and current values. At the end of each iteration, the values of the parameters are updated. Click models that use EM are much slower to train than ones that use MLE, but display better performance [57]. However, click models that use EM cannot be directly updated in online settings. When new user clicks become available, such models have to be retrained completely using all clicks observed so far. We address this problem and solve it by adopting online EM algorithms [122, 150]. To the best of our knowledge, no previous work has dealt with outdated clicks.

2.3 Method

In this section we formalize the online scenario for click models and discuss EM inference in this scenario. We propose *Online EM* to incorporate new click/skip observations into click models, and present a variant of the online EM algorithm to deal with outdated clicks/skips by discounting them over time (*EM with Forgetting*).

2.3.1 Online scenario and EM with Retraining

Various applications of click models (e.g., ranking, model-based evaluation, etc.) operate with click models trained using a historical click log. We assume that we have a click model M that is trained on n past observations of clicks/skips. As time passes, new clicks/skips are observed that can be used to update the trained model M. We would like to update M after observing m new clicks/skips, where $m \geq 1$. The task of online parameter estimation is, thus, to consider $n + m$ observations for training (or updating) the click model M.

Completely re-training the model parameters using EM inference has a computational complexity of $O([n+m] \cdot K)$, where K is the number of iterations. If we perform T consecutive updates of size m, then the complexity becomes $O([n + m \cdot T] \cdot K)$ for the T-th update. Thus, T consecutive updates, each considering m new observations, have a complexity of $O([n \cdot T + m \cdot \sum_{i=1}^{T} i] \cdot K) = O([n \cdot T + m \cdot T^2] \cdot K)$, which is quadratic in the number of updates. We refer to this approach as *EM with Retraining*.

2.3.2 Online EM

Online (or incremental) EM reduces the computational cost of EM with Retraining by updating the model parameters using only a part of the available data [122, 150]. Following this idea, we adapt EM inference for click models so as to update the models' parameters in online settings.

Most click models (and particularly those that use EM) consist of Bernoulli-distributed random variables $X \sim Bernoulli(\theta)$, where θ is a parameter corresponding to a random variable X. For example, most click models have an attractiveness random variable $A_{qd} \sim Bernoulli(\alpha_{qd})$ that is equal to 1 if, and only if, document d is attractive to a user given query q. In a number of click models, such as UBM, DBN, and CCM, this random variable is not observed and so its corresponding parameter α_{qd} has to be estimated using EM inference.

In general, the parameter θ of a Bernoulli-distributed random variable X is computed as the number of query sessions where $X = 1$, divided by the number of query sessions where X is defined:

$$\theta = \tfrac{1}{|\mathcal{S}_X|} \textstyle\sum_{s \in \mathcal{S}_X} P(X^{(s)} = 1 \mid \mathbf{C}^{(s)}), \tag{2.1}$$

where $\mathcal{S}_X$ is the set of query sessions where random variable X is defined, $X^{(s)}$ is the value of X in a particular query session s and $\mathbf{C}^{(s)}$ is the vector of clicks and skips observed in session s.

However, if a random variable X is not observed (e.g., A_{qd} is not observed in UBM, DBN and CCM), the probability $P(X^{(s)} = 1 \mid \mathbf{C}^{(s)})$ cannot be computed directly from clicks $\mathbf{C}^{(s)}$. Instead, this probability should be estimated based both on the observed clicks/skips $\mathbf{C}^{(s)}$ and the current values of all the parameters of a click model (we will denote these values as $\mathbf{\Psi}^{(k)}$, where k is the iteration counter). Then, the parameter θ should be updated iteratively as follows:

$$\theta^{(k+1)} = \tfrac{1}{|\mathcal{S}_X|} \textstyle\sum_{s \in \mathcal{S}_X} P(X^{(s)} = 1 \mid \mathbf{C}^{(s)}, \mathbf{\Psi}^{(k)}). \tag{2.2}$$

In an online scenario, we have a click model M that is trained on n past observations of clicks/skips. In other words, all the parameters of M are already estimated, i.e., the sum in Eq. 2.2 is calculated for all θ's. Let P_X denote this sum. Then, the learned parameter, denoted as $\theta^{(curr)}$, can be written as $\theta^{(curr)} = P_X/|\mathcal{S}_X|$. When we observe a new query session s with clicks $\mathbf{C}^{(s)}$, we need to update the current value $\theta^{(curr)}$ taking into account both the already calculated quantity P_X and the new observations $\mathbf{C}^{(s)}$. Iteratively re-estimating all the parameters of the model M for every newly observed query session is prohibitively slow in practice. Instead, we adapt the online EM inference method [122, 150] and update the value of the click model parameters as follows:

$$\theta^{(new)} = \frac{P_X + P(X^{(s)} = 1 \mid \mathbf{C}^{(s)}, \mathbf{\Psi}^{(k)})}{|\mathcal{S}_X| + 1}, \tag{2.3}$$

where the probability added to P_X is the same as in Eq. 2.2.

Eq. 2.3 allows us to update the model parameters instantly, moving from the current values $\theta^{(curr)}$ to the new values $\theta^{(new)}$ as soon as new clicks/skips $\mathbf{C}^{(s)}$ are observed. For this reason, we call this approach *Online EM*.

The complexity of Online EM is linear with respect to the number of observations, i.e., it is $O(m)$ for m new clicks/skips no matter how many updates are performed. T consecutive updates, each considering m new observations, have a complexity of $O(m \cdot T)$, which is linear with respect to the number of updates. Moreover, the complexity of this approach does not depend on the number of past observations n (which could have a significant impact on the execution time in case $n \gg m$) and the number of iterations K.

2.3.3 EM with Forgetting

Online EM addresses the problem of how to efficiently update click models in an online scenario by considering newly observed clicks/skips. The second challenge here is how to deal with past clicks, which may become outdated as time passes. In this section we adopt the idea of "forgetting" (or "aging") [102, 122] and propose the *EM with Forgetting* method that discounts past observations depending on their age. In particular, during every update of the model parameters, a certain percentage of past observations, denoted η here, can be "forgotten" as follows:

$$\theta^{(new)} = \frac{P_X \cdot (1-\eta) + P(X^{(s)} = 1 \mid \mathbf{C}^{(s)}, \mathbf{\Psi}^{(k)})}{|\mathcal{S}_X| \cdot (1-\eta) + 1}. \tag{2.4}$$

The forgetting ratio η has the following interpretation. Assume that after observing m new clicks/skips and performing m updates for a particular query-document pair, we need to forget $x\%$ of past observation. Then $(1-\eta)^m = 1-x$, so $\eta = 1 - \sqrt[m]{1-x}$. E.g., if we would like to forget 50% of past observations after 20 updates, then $\eta \approx 0.034$. To forget 10% after 10 updates, we need to set $\eta \approx 0.01$.

The computational complexity of this approach is the same as for Online EM, i.e., $O(m)$ for a single update of size m and $O(m \cdot T)$ for T consecutive updates.

2.4 Experimental Setup

In this section we describe the dataset and evaluation methodology that we use to test and analyze the efficiency and effectiveness of our proposed online EM methods for click models.

2.4.1 Dataset

There exist several publicly available datasets with click logs [34]. The one suitable for our task is the Yandex personalized web search challenge dataset (PWSC),[1] because it contains both the click/skip and time information. The timestamps provided are at the level of days. We use the first two weeks of the dataset, consisting of 33,310,079 query sessions, as historical data and the next 13 days, consisting of 31,862,774 query sessions, to simulate the incoming click stream.

2.4.2 Methodology

To evaluate our proposed online EM methods, we use the following experimental protocol:

1. Consider a click model M, trained on days $1, \dots, x$ of the click log. Initially, $x = 14$.
2. Evaluate M using the click log of day $x + 1$.
3. Update M using data collected on day $x + 1$.
4. Increment x and repeat steps 1–3 until all days of the dataset have been considered, i.e., until $x + 1 = 27$.

Since the online scenario for click models has not been studied yet, there are no standard baselines to compare against for online click modeling. To get an understanding of the relative efficiency and effectiveness of our online EM updating strategies, we compare them to two extreme cases:

(i) **Static**, where a click model that has been trained once is kept unchanged, i.e., step 3 of the above protocol is not performed. We expect that this strategy has the lowest effectiveness, because it does not keep click models up-to-date with the incoming click stream. The static strategy does not require any additional computations and so it has the highest efficiency.

(ii) **EM with Retraining**, where a click model is re-trained from scratch every day, using historical and newly observed click-through data, i.e., in step 3 of the above protocol instead of updating the model, we retrain it using all data from days $\{1, \dots, x + 1\}$. This strategy considers all available observations and does multiple passes over the data, so we expect it to have the highest effectiveness and we can treat it as an oracle; for the same reasons, it should have low efficiency.

[1] `https://www.kaggle.com/c/yandex-personalized-web-search-challenge`

Table 2.1: Performance of click models with EM inference trained on the first two weeks of the click log and evaluated on the next 13 days. $^{\triangle}$ denotes the best click model with statistically significant differences compared to other models.

Model	Log-likelihood	Perplexity
UBM	–0.2204$^{\triangle}$	1.2825$^{\triangle}$
DBN	–0.3523	1.3135
CCM	–0.3534	1.3176

2.4.3 Metrics

We are interested in assessing two aspects of our proposed strategies for updating click models: efficiency and effectiveness. We assess efficiency by the time it takes to update click models. And we measure effectiveness of our click model updating strategies using standard metrics, namely log-likelihood and perplexity [34, 46, 59]. Statistical significance of observed differences is determined using a paired Student t-test at the 0.01 level.

When measuring perplexity and log-likelihood, we use all available data to train and test click models. In this case, training is performed on a MapReduce cluster. To measure execution time, we use a part of the dataset that can be processed on a single machine with 2.00GHz CPU and 64Gb RAM (because we cannot control the cluster load and distribution of our jobs). In particular, we use the first 100K query sessions of each day to measure the time it takes to train and update click models.

2.4.4 Click models

To assess the impact of online EM on click models, we need to select one or more click models that require EM for their inference. DBN [27], UBM [46], and CCM [59] are the standard click models of this kind [34]. Table 2.1 shows performance of the above models when trained on the first two weeks of the PWSC dataset and evaluated on the next 13 days.

We see from Table 2.1 that UBM significantly outperforms both DBN and CCM, a finding that is in line with other results reported on the Yandex relevance prediction challenge dataset [34, 57].[2] For this reason, we use the UBM model in our experiments.

2.5 Results

In this section we evaluate the efficiency and effectiveness of our proposed online EM methods. We compare the time it takes to update click models using EM with Retraining and Online EM. We measure the effectiveness of Online EM and compare it to the oracle performance of EM with Retraining. We evaluate the effectiveness of EM with Forgetting using a range of forgetting rates η.

[2]See also `https://academy.yandex.ru/events/data_analysis/relpred2011/`

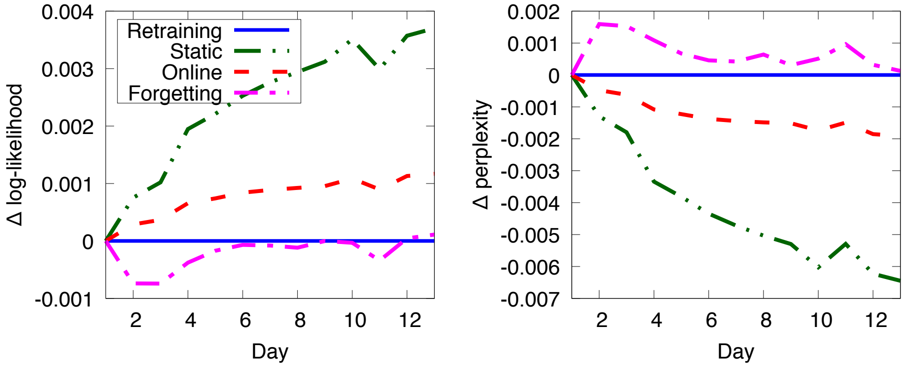
Table 2.2: Execution time in minutes of EM with Retraining, Online EM and EM with Forgetting when updating UBM on days 15–27 of the PWSC dataset. Time is measured on a subset of the click log (first 100K query sessions of each day).

Update strategy	**Min** (day 15)	**Max** (day 27)	**Average**	**Total**
EM with Retraining	28.1	49.6	39.7	516.1
Online EM	0.3	0.3	0.3	3.9
EM with Forgetting	0.3	0.3	0.3	3.9

2.5.1 Efficiency

In §2.3 we show that Online EM and EM with Forgetting have lower computational complexity than EM with Retraining. Here, we show that the gains from the reduced complexity are also practically important. To do that, we measure the time it takes for Online EM, EM with Forgetting and EM with Retraining to update the trained UBM model once a day using days 15–27 of the PWSC dataset. (As explained in the previous section, we use the first 100K query sessions of each day.)

Table 2.2 lists the execution times of EM with Retraining, Online EM, and EM with Forgetting. Online EM and EM with Forgetting are 130 times more efficient than EM with Retraining. This improvement can be explained by the fact that EM with Retraining does K passes over $n + m$ observations, while Online EM (and EM with Forgetting) does only 1 pass over m observations. Also, Online EM (and EM with Forgetting) requires constant time to incorporate m observations (18 seconds), while the time for EM with Retraining grows from 28 minutes (for day 15) to 50 minutes (for day 27).



(a) Log-likelihood (lower is better) (b) Perplexity (higher is better)

Figure 2.1: Difference in log-likelihood and perplexity between EM with Retraining (oracle) and other methods. EM with Forgetting uses $\eta = 0.001$.

Table 2.3: Average effectiveness of different update strategies. All update strategies are significantly better than the Static approach. $^{\triangle}/^{\triangledown}$ denote statistically significant differences with respect to EM with Retraining; $^{\blacktriangle}/^{\blacktriangledown}$ denote statistically significant differences with respect to Online EM.

Update strategy	**Log-likelihood**	**Perplexity**
EM with Retraining	–0.2180	1.2783
Static	–0.2204$^{\triangledown}$	1.2825$^{\triangledown}$
Online EM	–0.2188$^{\triangledown}$	1.2796$^{\triangledown}$
EM with Forgetting, $\eta = 0.001$	–0.2178 $^{\blacktriangle}$	1.2777$^{\triangle\blacktriangle}$
EM with Forgetting, $\eta = 0.005$	–0.2180 $^{\blacktriangle}$	1.2775$^{\triangle\blacktriangle}$
EM with Forgetting, $\eta = 0.01$	–0.2183$^{\triangledown\blacktriangle}$	1.2775$^{\triangle\blacktriangle}$

2.5.2 Effectiveness of Online EM

Next, we compare the effectiveness of Online EM, which incorporates newly observed click information on the fly, to the Static approach, which ignores newly observed clicks, and to EM with Retraining, which trains click models from scratch every day using historical data and newly observed clicks. In particular, we aim to answer the following questions:

RQ 1.1 Does Online EM achieve higher effectiveness compared to the Static approach?

RQ 1.2 Does Online EM achieve a comparable effectiveness to the one of EM with Retraining?

To answer these questions, we measure the log-likelihood and perplexity of the UBM model after daily updates using days 15–27 of the PWSC dataset.

We consider EM with Retraining as an oracle and plot the performance of the Static and Online EM approaches with respect to this oracle (Fig. 2.1). Note that the perplexity deltas are negative, because lower values of perplexity indicate higher effectiveness. In addition, we report the average values of log-likelihood and perplexity in Table 2.3 (top half).

Fig. 2.1 shows that EM with Retraining, which trains click models from scratch every day, achieves the best effectiveness. Online EM significantly improves the quality of click models over the Static approach, because it utilizes additional information, i.e., newly observed clicks/skips, to update the model parameters. However, the effectiveness of Online EM does not reach the effectiveness of EM with Retraining, while being close to it (see Table 2.3).

Note that, as opposed to Online EM, EM with Retraining first collects a number of click/skip observations and then uses all available information to update the model parameters. This way it uses more information than Online EM and, thus, achieves higher effectiveness but at substantially higher computational costs. In particular, EM with Retraining cannot be used to continuously update click models in online settings.

2.5.3 Effectiveness of EM with Forgetting

Here, we evaluate the effectiveness of EM with Forgetting, which deals with outdated click information. We aim to answer the following question:

RQ 1.3 Does EM with Forgetting, which discounts past click/skip observations, improve the effectiveness of Online EM, which does not do any discounting?

To answer this question, we measure the log-likelihood and perplexity of UBM that is updated using EM with Forgetting during days 15–27 of the PWSC dataset.

We use forgetting rates η of $0.001, 0.005$ and 0.01 in our experiments. Lower values of η did not result in any changes in effectiveness, while larger values did degrade the performance. The average absolute values of log-likelihood and perplexity are presented in Table 2.3 (bottom half), where all differences between EM with Forgetting and Online EM are statistically significant. The relative performance of EM with Forgetting ($\eta = 0.001$) compared to EM with Retraining is presented in Fig. 2.1.

First, we see that EM with Forgetting significantly improves the effectiveness of Online EM. This confirms our intuition that click information becomes outdated over time and past observations need to be discounted. Second, EM with Forgetting for $\eta = 0.001$ and $\eta = 0.005$ reaches the same effectiveness as EM with Retraining in terms of log-likelihood and significantly outperforms the latter in terms of perplexity for all forgetting rates. Hence, EM with Forgetting is at least as effective as EM with Retraining, while being orders of magnitude more efficient. This result suggests that in online scenarios EM with Forgetting should be preferred over both Online EM and EM with Retraining.

2.6 Conclusions and Future Work

We have studied click models in an online scenario, where a search engine deals with a stream of click/skip observations. Specifically, we answered the following research question:

RQ 1 How to keep click models up to date with changes in search engine algorithms and user preferences?

We have shown that the effectiveness of once trained click models degrades over time and, thus, these models must be constantly updated to keep up with the click stream. We have proposed Online EM to efficiently update click models on the fly using readily available EM equations as well as the EM with Forgetting method to deal with outdated click information by discounting past observations depending on their age. We have shown experimentally that the proposed methods are orders of magnitude more efficient than complete retraining and, as opposed to the latter, always keep click models up to date. We have also shown that Online EM and EM with Forgetting have significantly higher effectiveness than the static approach with no updates.

With this work we aim to move click modeling research closer to more real-world settings. As to future work, we plan to study the effect of the forgetting rate on different click model parameters; and to develop methods for dealing with outdated click information for other inference methods (e.g., probit).

3 Calibration

In the previous chapter, we trained click models using the default hyperparameters of the PyClick library.[1] As for many machine learning algorithms, the performance of click models strongly depends on the hyperparameters used for training. However, most authors do not provide sufficient details on how these hyperparameters are chosen.

We show that click models trained with suboptimal hyperparameters are prone to produce badly calibrated click probabilities. This means that their predicted click probabilities do not agree with the observed proportions of clicks in the held-out data. To repair this discrepancy we adapt a non-parametric calibration method called *isotonic regression*. We use a publicly available dataset to answer the following research question asked in §1.1:

RQ 2 Does calibration help to improve click model performance and make it less dependent on the choice of hyperparameters?

Our experimental results show that isotonic regression provides a good means to improve the performance of click models trained with suboptimal hyperparameters. And that the use of isotonic regression makes click models less sensitive to the choice of hyperparameters. Interestingly, the relative ranking of existing click models in terms of their predictive performance changes depending on whether or not their predictions are calibrated. We therefore advocate that calibration becomes a mandatory part of the click model evaluation protocol.

3.1 Introduction

Click models [34] are important and widely used tools for interpreting user behavior in web search. A common way to evaluate their performance is to measure how well they predict clicks on the documents presented on a SERP. As for many machine learning algorithms, the performance of click models strongly depends on the hyperparameters used for training.[2] However, most previous work on click models does not provide details on the choice of hyperparameters. Even in recent work by Grotov et al. [57],

This chapter is based on Borisov, Kiseleva, Markov, and de Rijke [19].

[1] `https://github.com/markovi/PyClick` (last visited August 16, 2018).

[2] This statement is based on our preliminary experiments with a range of click models and their hyperparameters. See §3.4 for details.

whose main purpose is to provide an objective evaluation and comparison of click models, there is no information about setting the hyperparameters of click models. In practice, tuning hyperparameters is time-consuming and might be prohibitively slow for complex models and large datasets. But it is well known [34] that click models may yield misleading results without properly tuned hyperparameters.

We hypothesize that click models trained with suboptimal hyperparameters are often not well *calibrated*. This means that their predicted click probabilities do not agree with the observed proportions of clicks in the held-out data. We validate how well a click model is calibrated using a reliability diagram [121]. For each rank, we split query sessions into $N = 100$ buckets according to the predicted click probabilities, where the i-th bucket corresponds to click probabilities in the range from $\frac{i}{N}$ to $\frac{i+1}{N}$, and plot the observed CTRs in these buckets. For a well-calibrated click model, the observed CTRs in each bucket should lie in the range of the predicted click probabilities associated with this bucket. Fig. 3.1 shows a reliability diagram of the click-chain model (CCM)

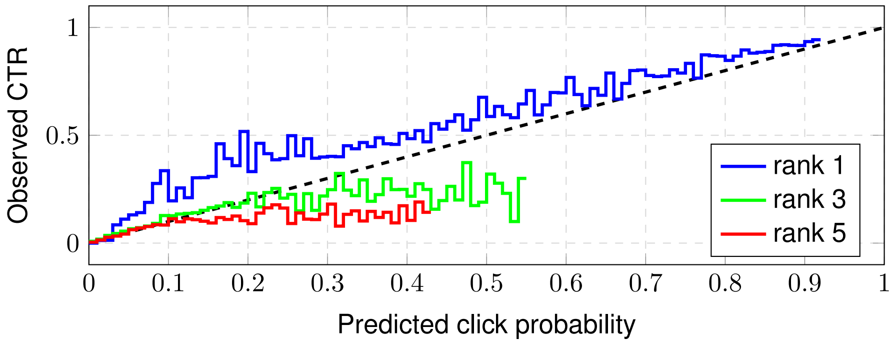


Figure 3.1: Reliability diagram of CCM at ranks 1, 3 and 5. (Best viewed in color.)

trained on a publicly available dataset released for the Yandex Relevance Prediction challenge (see §3.4 for details) with suboptimal hyperparameters.[3] We learn from Fig. 3.1 that CCM tends to underestimate click probabilities at rank 1 and overestimate click probabilities at ranks 3 and 5. We observe similar trends for other click models and therefore conclude that click models suffer from the issue of bad calibration.

There are two approaches to calibration: parametric and non-parametric [123]. The parametric approach is less flexible but also requires less data for calibration. The non-parametric approach is more general but requires more calibration data. Since real-world click logs are large, we follow the latter approach. In particular, we propose to use *isotonic regression* [137] to repair the discrepancy between the click probabilities predicted by a model and the proportion of clicks in the held-out data.

Our experimental results show that (i) isotonic regression significantly improves click models trained with suboptimal hyperparameters in terms of perplexity; and that (ii) calibrated click models are less sensitive to the choice of hyperparameters than the original (non-calibrated) ones. We also show that the relative ordering of existing click

[3]For rank 1, the prior values of the CCM parameters are distributed according to $\text{Beta}(1, 10)$; for ranks 3 and 5, according to $\text{Beta}(1, 2)$.

models by their predictive performance changes depending on whether or not their predictions are calibrated. Therefore, we advocate that calibration becomes a mandatory part of the click model evaluation protocol.

3.2 Background and Related Work

We discuss two types of related work: assessment of probabilities and click modeling.

3.2.1 Assessment of probabilities

Estimating probabilities of future events is important for effective decision making [103, 146]. Studies show that people are generally not good at these tasks [103]. They tend to overestimate or underestimate their confidence, which introduces biases in their predictions [103]. These biases can be reduced with proper training, called *calibration*.[4]

Recent work demonstrates that many popular machine learning methods also suffer from the issue of bad calibration [42, 123, 131, 175, 176]. Below we focus on work that examines calibration properties of different learning algorithms for binary classification. A multi-class classification problem can be reduced to a set of binary classification tasks, and, as Zadrozny and Elkan [176] show, the quality on the multi-class classification problem improves (in terms of *mean squared error* and *error rate*) when these binary classifiers are calibrated.

Niculescu-Mizil and Caruana [123] show that under unrealistic independence assumptions, Naive Bayes tends to produce probabilities that are too close to the extreme values of 0 and 1 (see [42] for a theoretical analysis), while SVM and boosted decision trees rarely predict probabilities that are close to 0 and 1. This suggests that predictions of these learning algorithms are biased. Zadrozny and Elkan [175] show that binning helps to improve the performance of Naive Bayes. They advocate the use of a special smoothing technique, called *curtailment*, to improve the calibration properties of decision trees. Niculescu-Mizil and Caruana [123] also suggest that neural networks and random forests tend to produce well-calibrated probabilities.

To alleviate the discrepancy between probabilities predicted by a binary classifier and observed frequencies, both parametric and non-parametric calibration methods have been investigated. Platt [131] puts forward the idea of fitting a sigmoid transformation between the outputs predicted by the binary classifier and the observed labels. Zadrozny and Elkan [176] suggest using isotonic regression [137], which learns a monotone transformation of the scores computed by the binary classifier to probabilities of class I. Niculescu-Mizil and Caruana [123] recommend using Platt scaling when the data used for calibration is limited and isotonic regression when there is enough data for calibration.

[4]According to Murphy and Winkler [121], experienced weather forecasters can quantify uncertainty in their predictions in a reliable manner.

3.2.2 Click modeling

Click data is a valuable signal for improving web search [27, 80]. However, accurately interpreting user clicks on a SERP is not straightforward due to the so-called *position bias effect* [58, 81]: people tend to click more on the documents presented on top positions than on the documents presented on lower positions. To account for this and other types of bias, click models have been proposed [34].

Traditional click models consist of Bernoulli-distributed random variables $X \sim \text{Bernoulli}(\theta)$ associated with query-document pairs [34]. Here, θ denotes a parameter associated with the query-document pair, e.g., *attractiveness* (i.e., the probability of a user examining the document's snippet) and *satisfactoriness*, (i.e., the probability of a user's information need being satisfied after interacting with the document). The value of the parameter θ is estimated during training. It is initially specified by a Beta distribution, $\theta_{\text{prior}} \sim \text{Beta}(\alpha, \beta)$, and then updated upon observing new click/skip data. The choice of the training hyperparameters α and β impacts the overall performance of click models, especially in query sessions that contain rare or previously unseen query-document pairs. Existing work on click models rarely provides sufficient details on tuning click model hyperparameters, even when they aim to systematically compare click models [57], which is troublesome because experiments without properly tuned hyperparameters may yield misleading results [34].

The key distinction of our work compared to the work listed above is that we are the first to improve the performance of click models by applying calibration, which also reduces the overhead of hyperparameters tuning.

3.3 Method

Click models are trained to predict probabilities of a user clicking on the ranked list of documents $d_1, \ldots, d_n$ returned by a search engine in response to a user's query q. Click models utilize different assumptions about how a user interacts with $d_1, \ldots, d_n$. E.g., many click models make the *linear traversal assumption* [36], which states that a user examines documents on a SERP from top to bottom. Such models predict the probability of observing a click on document d_{r+1} given a user's query q and clicks $c_1, \ldots, c_r$ on the higher ranked documents:

$$P(c_{r+1} = 1 \mid q, d_1, \ldots, d_{r+1}, c_1, \ldots c_r), \tag{3.1}$$

where $c_i = 1$ if a user clicked on document d_i, and 0 otherwise. In order to compare click models, which follow different assumptions about user click behavior, conditional click probabilities, presented in Eq. (3.1), are marginalized to click probabilities $P(c_{r+1} = 1 \mid q, d_1, \ldots, d_{r+1})$ that are unconditional on previous clicks:

$$\sum_{(c_1,\ldots,c_r)} P(c_{r+1} = 1 \mid q, d_1, \ldots, d_{r+1}, c_1, \ldots, c_r), \tag{3.2}$$

where the sum is computed over all possible click combinations on the first r documents.

As discussed in §3.1, for existing click models the click probabilities shown in Eq. (3.1) and Eq. (3.2) do not represent the observed CTRs well and, thus, need to be calibrated. We calibrate these probabilities separately for each rank. Following the recommendations in [123] and considering that real-world click data is usually available in large quantities, we adopt a non-parametric calibration method, namely isotonic regression [137]. For each rank r, isotonic regression learns a function $g_r(P)$ that adjusts the click probabilities predicted at rank r. Specifically, it solves the following optimization problem:

$$g_r^* = \arg\min_{g \in \mathcal{G}} \sum_{i=1}^{N} [g(P_r(s^i)) - c_r^i]^2, \tag{3.3}$$

where $\mathcal{G}$ denotes the set of all piecewise linear, isotonic (non-decreasing), continuous functions, N denotes the number of query sessions used for calibration, $P_r(s^i)$ denotes the predicted click probability at rank r in query session s^i and c_r^i denotes whether the user clicked on the document at rank r in query session s^i.

We use the pair-adjacent violators (PAV) algorithm [4] to find the optimal function $g_r^*(P)$. This is done in three steps, illustrated in Fig. 3.2. First, we sort query sessions s^i by the predicted click probabilities at rank r:

$$P_r(s^{i-1}) \leq P_r(s^i) \qquad \forall i = 2, \ldots, N. \tag{3.4}$$

We use red dots to display the output of this step in Fig. 3.2.

Second, we fit a piecewise linear function $g_r(P)$ to the sorted sequence of pairs $[P_r(s^1), c_r^1], \ldots, [P_r(s^N), c_r^N]$ in the following way. For P that is lower than $P_r(s^1)$, i.e., lower than the first click probability in the sorted sequence, $g_r(P)$ returns zero. For P that is larger than $P_r(s^N)$, i.e., larger than the last click probability in the sorted sequence, $g_r(P)$ returns the value of the last click c_r^N. For P that is between two consecutive $P_r(s^{i-1})$ and $P_r(s^i)$, $g_r(P)$ returns the value of click c_r^{i-1}. Formally, this can be written as follows:

$$g_r(P) = \begin{cases} 0 & P < P_r(s^1) \\ c_r^{i-1} & P \in [P_r(s^{i-1}), P_r(s^i)) \quad \forall i = 2, \ldots, N \\ c_r^N & P \geq P_r(s^N). \end{cases} \tag{3.5}$$

The piecewise linear function $g_r(P)$ calculated at this step is shown with the blue line in Fig. 3.2.

Third, if the above $g_r(P)$ is not isotonic, there exist two consecutive query sessions s^{i-1} and s^i for which $g_r(P)$ decreases (instead of increasing or staying constant), i.e., $g_r(P_r(s^{i-1})) > g_r(P_r(s^i))$. Such query sessions are called *pair-adjacent violators*. In this case, we change the value of $g_r(P)$ for the interval $P \in [P_r(s^{i-1}), P_r(s^{i+1}))$ to the average of $g_r(P_r(s^{i-1}))$ and $g_r(P_r(s^i))$.[5] This way, for pair-adjacent violators $g_r(P)$ does not decrease anymore, but stays constant and equal to the above-mentioned average. This averaging process is performed in the direction from $P_r(s^1)$ to $P_r(s^N)$. In the end, $g_r(P)$ becomes isotonic as shown with the green line in Fig. 3.2. See [4] for a proof of the optimality of $g_r(P)$ with respect to Eq. (3.3).

[5] If $i = N$, we perform this averaging for $P \geq P_r(s^{N-1})$.

Note that the optimal calibration function $g_r^*(P)$ learned by PAV outputs a probability of 0 for $P < P_r(s^1)$ and might output a probability of 1 for large values of P. Following [123, 131] and to avoid problems with taking the log of $g_r^*(P)$, we trim $g_r^*(P)$ to predict probabilities in the range $[\delta, 1-\delta]$ instead of the range $[0,1]$. In our experiments, we use $\delta = 0.01$.

Now that we have learned a calibration function $g_r^*(P)$ for a click model, the calculation of click probabilities at rank r works as follows. Given a query session s, the click model predicts the click probability $P_r(s)$ for rank r in that session. This could be either the conditional, Eq. (3.1), or unconditional, Eq. (3.2), probability.[6] The predicted probability $P_r(s)$ is then passed to the calibration function $g_r^*(P)$, which outputs the calibrated probability. This calibrated probability is then used for click prediction.

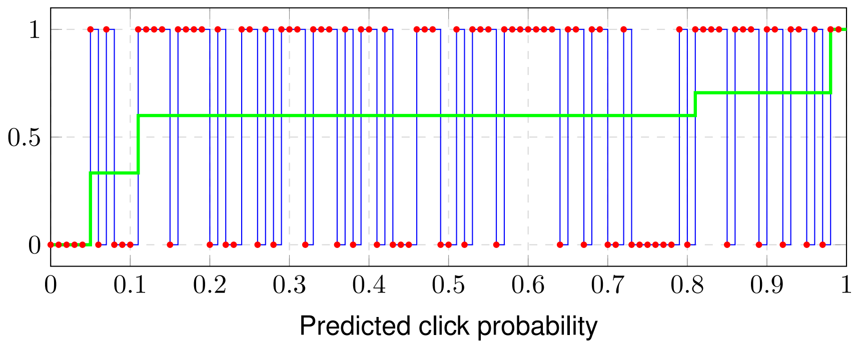


Figure 3.2: Illustration of the three steps of the PAV algorithm. Red dots represent observed clicks vs. predicted click probabilities. The blue line represents the piece-wise linear function in Eq. (3.5). The green line represents the learned isotonic transformation from original click probabilities to calibrated click probabilities. (Best viewed in color.)

3.4 Experimental Setup

In this section we describe our experimental setup. We outline the research questions in §3.4.1. Our dataset and evaluation methodology are described in §3.4.2 and §3.4.3. In §3.4.4 we describe the experiments we conduct to answer our research questions.

3.4.1 Research questions

We split RQ 2 into two subquestions:

RQ 2.1 Does the calibration method presented in §3.3 help to improve the performance of existing click models?

RQ 2.2 Does the calibration method presented in §3.3 make click models less dependent on the choice of hyperparameters?

[6]Note that calibration should be done separately for each of those probabilities.

3.4.2 Dataset

We conduct our experiments using a publicly available dataset released for the Yandex Relevance Prediction challenge by Yandex, the major search engine in Russia.[7] Query sessions are ordered by time. We use the first 1,000,000 query sessions as the *training set*, the following 100,000 query sessions as the *development set* and the next 100,000 query sessions as the *test set*.

3.4.3 Evaluation methodology

We evaluate click models using perplexity [34], which measures how "surprised" a model is upon observing a particular set of clicks on a SERP. We calculate perplexity at position k as follows:

$$\text{Perplexity@k} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(c_k=c_k^i|q^i,d_1^i,\ldots,d_n^i)}, \tag{3.6}$$

where N denotes the number of query sessions in the test set; q^i is a query in the i-th session; $d_1^i,\ldots,d_n^i$ are documents retrieved by a search engine in the i-th session in response to the query q_i; $c_k^i = 1$ if a user clicked on the document and 0 otherwise. Following [46], we use perplexity averaged over all positions as our main metric. Lower values of perplexity correspond to higher quality of a model. For each click model, we perform significance testing using a paired t-test on the perplexity scores computed using different sets of hyperparameters. Differences are considered statistically significant for p-values lower than 0.05. We do not evaluate click models on the relevance prediction task [34], because the inferred query-document-specific parameters used for ranking are not affected by the proposed calibration method.

3.4.4 Experiments

We design our experiments to answer the research questions stated in §3.4.1.

Experiment 1. To answer RQ 2.1, we measure, before and after calibration, the average of the perplexity values computed for a click model $\mathcal{M}$ trained with different hyperparameters. If the average value of perplexity is lower after calibration, we conclude that calibration helps to improve the performance of $\mathcal{M}$. Otherwise, we conclude that calibration does not help or even hurts the performance of $\mathcal{M}$.

Experiment 2. To answer RQ 2.2, we measure, before and after calibration, the variance of the perplexity values computed for a click model $\mathcal{M}$ trained with different hyperparameters. If the variance of the perplexity values is lower after calibration, we conclude that calibration makes $\mathcal{M}$ less dependent on the choice of hyperparameters. Otherwise, we conclude that calibration does not make $\mathcal{M}$ less dependent on the choice of hyperparameters or even makes it more sensitive to the choice of hyperparameters.

We conduct our experiments using four PGMs that are often used for modeling and predicting clicks on a SERP: the dynamic Bayesian network (DBN) [27], the dependent click model (DCM) [60], the click-chain model (CCM) [60], and the user browsing

[7] `http://imat-relpred.yandex.ru/en/datasets` (last visited August 16, 2018).

model (UBM) [46]. We train these click models by maximizing the likelihood of the observed click/skip events in our logs. For DCM we optimize the likelihood directly, and for DBN, CCM and UBM we use the EM algorithm with 50 iterations. We set the prior values of the parameters of these click models to $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{5}$, $\frac{1}{6}$, $\frac{1}{7}$, $\frac{1}{8}$, $\frac{1}{9}$, $\frac{1}{10}$, $\frac{2}{3}$, $\frac{2}{4}$, $\frac{2}{5}$, $\frac{2}{6}$, $\frac{2}{7}$, $\frac{2}{8}$, $\frac{2}{9}$, $\frac{2}{10}$, $\frac{3}{4}$, $\frac{3}{5}$, $\frac{3}{6}$, $\frac{3}{7}$, $\frac{3}{8}$, $\frac{3}{9}$, $\frac{3}{10}$.

3.5 Results

In this section, we present the outcomes of the experiments described in §3.4.4 and provide answers to the research questions stated in §3.4.1.

3.5.1 Experiment 1

Table 3.1 shows the perplexity of the click models CCM, DBN, DCM and UBM averaged over ranks and over runs with different hyperparameters. The rows of Table 3.1 correspond to:

1. Baseline w/o dev set, a method where click models are trained on the training set;

2. Baseline w/ dev set, a method where click models are trained on the union of the training set and the development set;

3. Calibrated, a method where click models are trained on the training set and calibrated on the development set.

Table 3.1: Perplexity of click models with and without calibration averaged over ranks and over runs with different hyperparameters. Improvements of the proposed calibration method over both baselines are statistically significant ($p < 0.001$). The best results are given in bold.

		Average perplexity			
#	Method	CCM	DBN	DCM	UBM
1	Baseline w/o dev set	1.3896	1.3915	1.3826	1.3724
2	Baseline w/ dev set	1.3890	1.3908	1.3822	1.3719
3	**Calibrated**	**1.3722**	**1.3705**	**1.3752**	**1.3659**

From Table 3.1, we conclude that isotonic regression improves the performance of the selected click models. The differences in performance between the click models trained (i) on the training set only, and (ii) on the union of the training set and the development set (Table 3.1, row 2 vs. row 1) are much less than the gains achieved from using the development set for calibration (Table 3.1, row 3 vs. row 1). This means that the gains obtained from the calibration method described in §3.3 are not (only) due to using more data (i.e., the development set), but are due to fixing the miscalibration problem.

Interestingly, the relative ranking of click models in terms of perplexity differs, depending on whether we use calibration or not. From Table 3.1, we infer the following rankings:

$$\text{UBM} > \text{DCM} > \text{CCM} > \text{DBN} \qquad \text{(w/o calibration)} \tag{3.7}$$

$$\text{UBM} > \text{DBN} > \text{CCM} > \text{DCM} \qquad \text{(w/ calibration)} \tag{3.8}$$

where we write $\mathcal{M}_X > \mathcal{M}_Y$ to denote that click model $\mathcal{M}_X$ has better prediction performance (in terms of perplexity) than click model $\mathcal{M}_Y$. Intuitively, the results w/ calibration make more sense, because DBN and CCM make more realistic assumptions than DCM, which assumes (i) that a user's information need cannot be satisfied directly on a SERP (i.e., a user needs to click at least one document presented on the SERP); and (ii) that the probability of examining the document at rank $(r+1)$ after clicking on the document presented at rank r depends solely on the rank r and not on the relevance of the document presented at rank r.

Answering RQ 2.1, we conclude that the calibration method described in §3.3 provides a good means to fix the miscalibration problem and, as a result, allows us to improve the performance of existing click models on the standard click prediction task.

3.5.2 Experiment 2

Tables 3.2 and 3.3 show the empirical variance in the perplexity values computed for click models trained with different sets of hyperparameters. Table 3.2 lists the absolute values; Table 3.3 lists the percentages w.r.t. the baseline w/o dev set. For the methods, we use the same naming conventions as in Table 3.1. We find that the

Table 3.2: The empirical variance in the perplexity values computed for click models trained with different sets of hyperparameters. The best results are given in bold.

		100 × variance in perplexity			
#	Method	CCM	DBN	DCM	UBM
1	Baseline w/o dev set	0.1025	0.1113	0.0908	0.0629
2	Baseline w/ dev set	0.1001	0.1087	0.0886	0.0606
3	**Calibrated**	**0.0079**	**0.0179**	**0.0619**	**0.0011**

Table 3.3: The empirical variance in the perplexity values computed for click models trained with different sets of hyperparameters. The best results are given in bold.

		Variance in perplexity			
#	Method	CCM	DBN	DCM	UBM
1	Baseline w/o dev set	100%	100%	100%	100%
2	Baseline w/ dev set	97.66%	97.66%	97.58%	96.34%
3	**Calibrated**	**7.71%**	**16.08%**	**68.17%**	**1.75%**

calibration method described in §3.3 reduces the variance in the perplexity values by 33.87%–98.25%, depending on the click model.

Answering RQ 2.2, we conclude that the calibration method described in §3.3 makes click models less dependent on the choice of hyperparameters.

3.6 Conclusions and Future Work

We introduced the notion of calibration in the context of click modeling and showed empirically that existing click models are prone to produce poorly calibrated predictions. Answering the main research question in this chapter,

RQ 2 Does calibration help to improve click model performance and make it less dependent on the choice of hyperparameters?

we concluded that calibration, namely isotonic regression, (i) improves the performance of click models, and (ii) makes click models less sensitive to tuning of hyperparameters. Therefore, we advocate that calibration becomes a mandatory part of the click model evaluation protocol. In future work, we are planning to incorporate calibration at training time, e.g., by means of hierarchical priors [163] or variational auto-encoders [91].

4 A Neural Click Model

In Chapters 2 and 3, we studied click models based on the *probabilistic graphical model* (PGM) framework, in which user behavior is represented as a sequence of observable and hidden events. The PGM framework provides a mathematically solid way to reason about a set of events given some information about other events. But the structure of the dependencies between the events has to be set manually. Different click models use different hand-crafted sets of dependencies (represented as PGMs), while all of them are, by necessity, simpifications.

We propose an alternative based on the idea of distributed representations: to represent the user's information need and the information available to the user with a *vector state*. The components of the vector state are learned to represent concepts that are useful for modeling user behavior. And user behavior is modeled as a sequence of vector states associated with a query session: the vector state is initialized with a query, and then iteratively updated based on information about interactions with the search engine results. This approach allows us to directly understand user browsing behavior from click-through data, i.e., without the need for a predefined set of rules as is customary for PGM-based click models.

We illustrate our approach using a set of *neural click models* and answer the following research question asked in §1.1:

RQ 3 How to design a neural network that would be able to learn patterns in user click behavior directly from logged interaction data?

Our experimental results show that the *neural click model* that uses the same training data as traditional PGM-based click models, has better performance on the click prediction task (i.e., predicting user click on search engine results) and the relevance prediction task (i.e., ranking documents by their relevance to a query). An analysis of the best performing neural click model shows that it learns similar concepts to those used in traditional click models, and that it also learns other concepts that cannot be designed manually.

This chapter is based on Borisov, Markov, de Rijke, and Serdyukov [17].

4.1 Introduction

Understanding users' interaction behavior with a complex Information Retrieval (IR) system is key to improving its quality. In web search, the ability to accurately predict the behavior of a particular user with a certain information need, formulated as a query, in response to a search engine result page allows search engines to construct result pages that minimize the time that it takes users to satisfy their information needs, or increase the probability that users click on sponsors' advertisements.

Recently, many models have been proposed to explain or predict user behavior in web search; see [34] for an overview. These models, also called *click models* as the main observed user interaction with a search system concerns clicks, are used for click prediction and they may help in cases where we do not have real users to experiment with, or prefer not to experiment with real users for fear of hurting the user experience. Click models are also used to improve document ranking (i.e., infer document relevance from clicks predicted by a click model) [27, 45], improve evaluation metrics (e.g., model-based metrics) [28, 32, 174] and to better understand a user by inspecting the parameters of click models [46].

Existing click models are based on the *probabilistic graphical model* (PGM) framework [95], in which user behavior is represented as a sequence of observable and hidden events such as clicks, skips and document examinations. The PGM framework provides a mathematically solid way to reason about a set of events given information about other events. The structure of the dependencies between the events has to be set manually. Different click models use different hand-crafted sets of dependencies (represented as probabilistic graphical models), while all of them are, by necessity, simplifications and likely to miss key aspects of user behavior.

We propose an alternative to the PGM-based approach—the *distributed representation* (DR) approach—in which user behavior is represented as a sequence of *vector states* that capture the user's information need and the information consumed by the user during search. These vector states can describe user behavior from more angles than the binary events used in PGM-based models (such as whether a user examined a document, or whether a user is attracted by a document), which makes them attractive for learning more complex patterns of user behavior than those hard-coded in existing click models.

We illustrate the distributed representation-based approach using a set of *neural click models*, and compare them against traditional PGM-based click models on a click prediction task (i.e., predicting user clicks on search engine results) and a relevance prediction task (i.e., ranking documents by their relevance to a query). Our experimental results show (i) that the neural click model that uses the same training data as traditional PGM-based click models has better performance on both the click prediction task and the relevance prediction task than the PGM-based click models; and (ii) that the performance of this neural click model can be further improved by incorporating behavioral information over all query sessions that is (a) generated by a particular query, and (b) contains a particular document. We also conduct an analysis of the model's internal workings, which shows that our neural click models learn concepts such as "the current document rank" and "the distance to the previous click," which are used in the *user browsing model* [46], a state of the art PGM-based click model. We also show that the neural click models learn other concepts that cannot be designed manually.

The main contribution of our work is the introduction of a distributed representation-based approach for modeling user behavior and several neural click models, which learn patterns of user behavior directly from interaction data, unlike conventional PGM-based click models that require these patterns to be set manually.

4.2 Related Work

We discuss two main types of related work: modeling click behavior of web search users and learning distributed representations of concepts.

4.2.1 Click models

Existing click models are based on the *probabilistic graphical model* (PGM) framework [95], in which user behavior is represented as a sequence of observable and hidden events such as clicks, skips and document examinations. Most probabilistic models of user behavior distinguish between two events (usually assumed independent): a document is *examined* by a user (E_d) and a document is *attractive* to a user (A_d).[1] Furthermore, most models make the *examination hypothesis* that a user clicks on a document ($C_d = 1$) if, and only if, she examined the document ($E_d = 1$) and was attracted by it ($A_d = 1$).[2] The examination probability is modeled differently by different click models, while the attractiveness probability is usually modeled with a parameter $\alpha_{q,d}$ that depends on a query and a document.

The *cascade model* (CM) [36] assumes that a user scans a *search engine result page* (SERP) from top to bottom until she finds a relevant document on which she clicks. In its canonical form, CM postulates that "a user who clicks never comes back, and a user who skips always continues," which limits its applicability to query sessions with exactly one click. This problem has been addressed in the *user browsing model* (UBM) [46], the *dynamic Bayesian network* (DBN) model [27], *dependent click model* (DCM) [60] and *click chain model* (CCM) [59] that use CM as their backbone. UBM introduces a set of examination parameters $\gamma_{r,r-r'}$ $(0 \leq r' < r \leq 10)$, and defines the examination probability of a document at rank r given that the previous click was at rank r' as $\gamma_{r,r-r'}$ ($r' = 0$ if none of the documents above r were clicked). DBN introduces a continuation parameter γ and per query-document pair *satisfactoriness* parameters $\beta_{q,d}$ that describe the *actual relevance* of the document d to the query q as opposed to the *attractiveness* parameters $\alpha_{q,d}$ that describe the *perceived relevance* of the document d to the query q. The examination probability of a document d is defined as the probability that a user was not satisfied by the documents ranked higher than d multiplied by the continuation parameter γ.

Recently, a wide range of click models have been proposed that exploit additional information, e.g., information about the user, her current task, the result presentation, the content of results, and other search characteristics. These include the *personalized click*

[1]The two-stage model also assumes that there is a skimming event prior to examination [109].

[2]Some recent click models [31, 158] relax this assumption to account for noise in user clicks: a user might click on an unattractive document or skip an attractive document because of carelessness.

model [144], the *task-centric click model* [179], *intent-aware* modifications of UBM and DBN [33], the *federated click models* [30], the *vertical-aware click model* [156], the *content-aware click model* [158], and *noise-aware* modifications of UBM and DBN [31].

Our work differs from the work discussed above in two important respects. First, we model user browsing behavior on a SERP as a sequence of vectors. Such a representation allows us to describe user behavior from more angles than a sequence of predefined binary events, which is common in existing click models. Second, our approach does not require a manually designed set of rules that describes user browsing behavior on a SERP—such rules constitute the key ingredient of existing click models based on probabilistic graphical models. Instead, we learn such rules directly from past user sessions, which allows us to capture more complex patterns of user behavior than the ones that are set manually.

4.2.2 Distributed representations of concepts

The idea of modeling behavioral phenomena as an emergent process of interconnected network activities of simple units was originally introduced by cognitive scientists [47], and later developed into the *parallel distributed processing* approach [138], which forms the basis of the artificial neural networks used in image recognition [97], speech recognition [56], machine translation [148] and other fields [8]. The main idea is to represent the input data with one or many vectors, whose components (which may not be interpretable alone) work together to represent concepts that are useful in the task under consideration; these vectors are called *distributed (vector) representations* of the input data.

Recent work that demonstrates the importance of learning distributed representations is due to Mikolov et al. [118]; the authors represent words with vectors and train a neural network to predict the vector of a word given the vectors of surrounding words. The resulting word-specific vectors capture many linguistic regularities, which make them useful for a broad range of applications.

In neural image processing, distributed representations of images are learned directly from image pixels. These image-specific vectors capture many visual regularities, which make them useful for image classification [97]. In neural speech recognition, distributed representations of audio signals are learned from the raw audio data, and then converted into sequences of words [56]. In neural machine translation, a sequence of words in one language is first transformed into an internal distributed representation and then converted into a sequence of words in another language [148].

Different applications use different network architectures to construct effective representations of their data. *Convolutional neural networks* (CNN) are used in image processing to abstract from the exact pixel locations and generalize to unseen images with similar pixel configurations [97]. *Recurrent neural networks* (RNN) are used in language modeling [117], speech recognition [56] and machine translation [148] to process word sequences of arbitrary length.

In this work we introduce distributed representations of user information needs and search results for modeling user browsing behavior in web search. To the best of our knowledge this is the first time such representations have been developed for this purpose.

4.3 Method

Below, we propose a general *neural click model framework*, in which user behavior is modeled as a sequence of distributed representations of the user's information need and the information consumed by the user during search. The principal advantage of the proposed framework over existing click models is that patterns of user behavior can be learned directly from interaction data, which allows us to capture more complex patterns of user behavior than the ones that are hard-coded in existing click models.

4.3.1 Neural click model framework

We model user browsing behavior in web search as a sequence of vector states ($\mathbf{s}_0$, $\mathbf{s}_1, \mathbf{s}_2, \ldots$) that describes the information consumed by the user as it evolves within a query session, i.e., $\mathbf{s}_{r-1}$ denotes the information consumed by the user before examining document d_r at rank r. We initialize the vector state $\mathbf{s}_0$ with a user query q, and iteratively update a vector state $\mathbf{s}_r$ to the vector state $\mathbf{s}_{r+1}$ based on user interactions i_r and the next document d_{r+1}:

$$\mathbf{s}_0 = \mathcal{I}(q), \tag{4.1}$$

$$\mathbf{s}_{r+1} = \mathcal{U}(\mathbf{s}_r, i_r, d_{r+1}). \tag{4.2}$$

We learn the mappings $\mathcal{I}(\cdot)$ and $\mathcal{U}(\cdot)$ to produce vector states $\mathbf{s}_0$, $\mathbf{s}_1$, $\mathbf{s}_2$, ... that are useful for predicting user clicks on a SERP. We predict the probability of a click on document d_{r+1} ($C_{r+1} = 1$) given the query q, interactions $i_1, \ldots, i_r$ and documents $d_1, \ldots, d_{r+1}$ from the vector state $\mathbf{s}_{r+1}$ using a function $\mathcal{F}(\cdot) \mapsto [0, 1]$:

$$P(C_{r+1} = 1 \mid q, i_1, \ldots, i_r, d_1, \ldots, d_{r+1}) = \mathcal{F}(\mathbf{s}_{r+1}). \tag{4.3}$$

Overall, the process of modeling user browsing behavior on a SERP can be unfolded in the following steps. See Figure 4.1 for an illustration.

1. (a) A user starts a search session by issuing a query q.
   (b) The vector state is initialized with q: $\mathbf{s}_0 = \mathcal{I}(q)$;
   the previous interactions are empty: $i_0 = \varnothing$.
2. (a) The user examines document d_1.
   (b) The vector state is updated with the examined document and the previous interactions: $\mathbf{s_1} = \mathcal{U}(\mathbf{s}_0, i_0, d_1)$.
3. (a) The user clicks on the examined document d_1 with probability $\mathcal{F}(\mathbf{s}_1)$ and skips it with probability $1 - \mathcal{F}(\mathbf{s}_1)$.
   (b) The user interactions i_1 are set using the information about the observed user interactions with d_1.
4. The user continues examining documents at ranks $r > 1$ repeating steps 2 and 3, where the indices 0 and 1 are replaced with $r - 1$ and r, respectively.

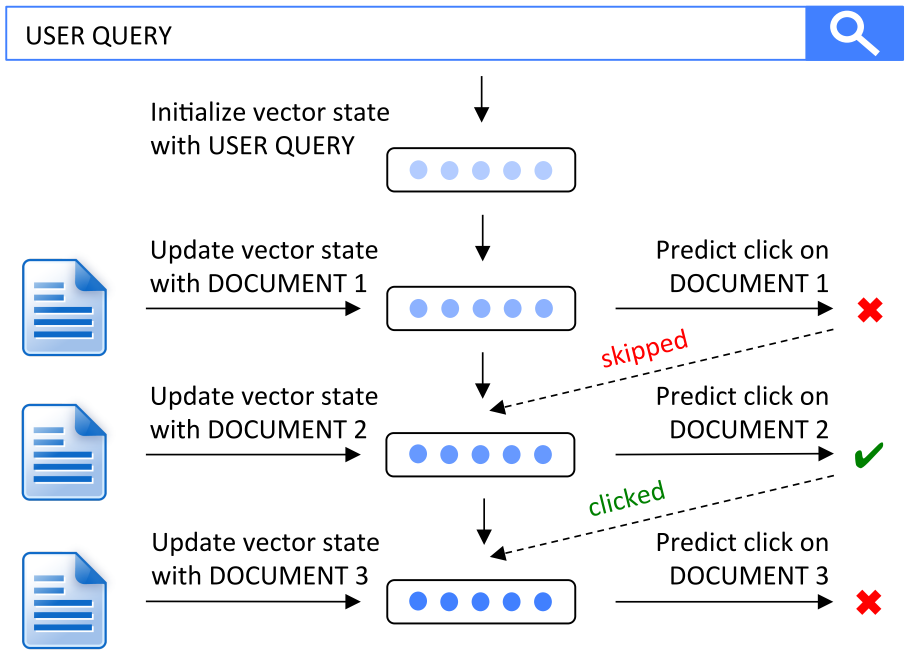

USER QUERY
Initialize vector state with USER QUERY
Update vector state with DOCUMENT 1
Predict click on DOCUMENT 1
skipped
Update vector state with DOCUMENT 2
Predict click on DOCUMENT 2
clicked
Update vector state with DOCUMENT 3
Predict click on DOCUMENT 3

Figure 4.1: Modeling user browsing behavior on a SERP in the neural click model framework.

Notice that similar to many existing click models, we do not explicitly model search abandonment (the user stops examining the SERP), but we train our model to predict low click probabilities for documents that are unlikely to be examined by the user.

The above is a general framework for modeling user browsing behavior on a SERP. In fact, most existing click models can be described by the mappings $\mathcal{I}(\cdot), \mathcal{U}(\cdot)$ and the function $\mathcal{F}(\cdot)$; see appendix (§4.7) for the descriptions of $\mathcal{I}(\cdot), \mathcal{U}(\cdot)$ and $\mathcal{F}(\cdot)$ used in DBN and UBM. However, our approach differs from UBM and DBN in an important aspect. The mappings $\mathcal{I}(\cdot)$ and $\mathcal{U}(\cdot)$ in UBM and DBN have no parameters that can be tuned during training, which means that all relevant information for predicting clicks on the next documents is included or discarded manually (according to the rules specified when designing the probabilistic graphical model). In our approach, these mappings are learned to collect information that is useful for predicting clicks on the next documents.

Now that we have specified our general approach to modeling user browsing behavior on a SERP, we need to detail how we represent the query q, the interactions i_r and the document d_{r+1} (§4.3.2), and how we learn the mappings $\mathcal{I}(\cdot), \mathcal{U}(\cdot)$ and the function $\mathcal{F}(\cdot)$ (§4.3.3).

4.3.2 Representations of queries, documents and interactions

We describe three sets of representations for a query q, a document d and interactions i.[3] Set 1 (QD) operates on the basis of query-document pairs similarly to traditional click models (e.g., DBN, UBM). Set 2 (QD+Q) extends Set 1 by considering all query sessions generated by the query q (including ones whose SERPs do not contain the document d). Set 3 (QD+Q+D) extends Set 2 by considering all query sessions whose SERPs contain the document d (including ones generated by queries other than the query q).

Set 1 (QD)

The first set of representations operates at the level of query-document pairs. It uses a trivial representation $\mathbf{q}^{(1)}$ of the query q, a query-dependent representation $\mathbf{d}^{(1)}$ of the document d and a click-based representation $\mathbf{i}^{(1)}$ of the interactions i.

Represention $\mathbf{q}^{(1)}$. We represent the query q with a zero vector of size 1.

Represention $\mathbf{d}^{(1)}$. We represent the document d by historical user interactions, observed on SERPs generated by the query q and containing the document d. User interactions on a SERP can be described in a number of ways: by clicks, dwell times, mouse movements, etc. We describe them by *click patterns*—the sets of documents that received a click—because clicks are the most widely logged form of user interaction. For example, the clicks on the first and third positions define the click pattern $[1, 0, 1, 0, 0, \ldots]$; the click on the second position defines the click pattern $[0, 1, 0, 0, \ldots]$.

Since there are $2^{10} = 1024$ possible click patterns and a document can be presented on any of the 10 positions, we represent the document d with a vector of size of 10240. In each component of the vector, we store the number of times a particular click pattern was observed on SERPs generated by the query q when presenting the document d at a particular rank.

Represention $\mathbf{i}^{(1)}$. We represent interactions i with a binary vector of size 1: the value 0 denotes the fact that a user skipped the previous document, the value 1 denotes the fact that a user clicked on the previous document.

Set 2 (QD+Q)

The second set of representations extends Set 1 by considering all query sessions generated by the query q. This is useful in cases when query sessions generated by the query q have different SERPs, which happens for various reasons: a change in the global ranking algorithm, due to a document index update or due to personalization. In particular, the representations used in Set 1 ignore the potentially useful information about user interactions in query sessions generated by the query q, whose SERPs do not contain the document d. Set 2 aggregates information about all query sessions generated by the query q in a representation $\mathbf{q}^{(2)}$. The representations of the document d and the interactions i stay the same as in Set 1, i.e., $\mathbf{d}^{(1)}$ and $\mathbf{i}^{(1)}$ respectively.

Represention $\mathbf{q}^{(2)}$. We represent the query q by click patterns, observed on SERPs generated by the query q. Since there are $2^{10} = 1024$ possible click patterns, we

[3]We drop the indices in d_{r+1} and i_r to simplify the notation.

represent the query q with a vector of size 1024. In each component of the vector $\mathbf{q}^{(2)}$, we store the number of times a particular click pattern was observed on SERPs generated by the query q.

Set 3 (QD+Q+D)

The third set of representations extends Set 2 by considering all query sessions whose SERPs contain the document d. This is particularly useful for rare queries, as it allows us to collect behavioral information over a larger number of observations than only considering query sessions generated by the query q, as done in $\mathbf{d}^{(1)}$. This behavioral information can be biased due to the fact that SERPs containing document d can be generated by queries with different intents.

However, most documents are presented in query sessions generated by queries with the same or similar intents. And even the behavioral information collected over query sessions generated by queries with different intents tells, e.g., about global attractiveness of document d, which might be useful for explaining so-called *sudden interest* clicks on a SERP. Thus, Set 3 extends the representation $\mathbf{d}^{(1)}$ by including information about all query sessions containing the document d (which we refer to as a representation $\mathbf{d}^{(3)}$ of the document d). To sum up, Set 3 uses the $\mathbf{q}^{(2)}$ representation of query q, the concatenation of the $\mathbf{d}^{(1)}$ and $\mathbf{d}^{(3)}$ representations of document d and the $\mathbf{i}^{(1)}$ representation of the interactions i.

Representation $\mathbf{d}^{(3)}$. We represent document d by click patterns observed on SERPs that contain the document d (but not necessarily generated by the query q). Since a document can be presented on any of the 10 positions and there are $2^{10} = 1024$ possible click patterns, we represent the document d with a vector of size 10240. In each component of the vector, we store the number of times a particular click pattern was observed on all SERPs containing the document d at a particular rank.

4.3.3 Implementations of $\mathcal{I}$, $\mathcal{U}$ and $\mathcal{F}$

Now that we have described the representations that we aim to use, we need to detail how we implement the mappings $\mathcal{I}(\cdot)$, $\mathcal{U}(\cdot)$ and the function $\mathcal{F}(\cdot)$ that form a key ingredient of our neural click model framework (§4.3.1). Following the literature on recurrent neural networks, we propose two configurations: the *RNN configuration* and the *LSTM configuration*. And then we describe how to learn the parameters of these configurations.

Below, we use the following notation: we write $\mathbf{q}$ to denote the vector representation of the query q, $\mathbf{d}_r$ to denote the vector representation of the document d_r, $\mathbf{i}_r$ to denote the vector representation of the interactions i_r and $\mathbf{c}_r$ to denote the vector of size 1, whose single component equals $P(C_r = 1 \mid q, i_1, \ldots, i_{r-1}, d_1, \ldots, d_r)$.

RNN configuration

The RNN configuration is illustrated in Figure 4.2(a). It uses a fully-connected layer[4] to initialize the vector state $\mathbf{s}_0$ and a simple recurrent connection to propagate the

[4]We refer the reader to [8] for explanations and background material on neural networks.

$$\begin{tikzcd}
\mathbf{q} \arrow[r, "\mathbf{W}_{qs}"] & \mathbf{s}_0 \arrow[d, "\mathbf{W}_{ss}"] & \\
\mathbf{0}_i, \mathbf{d}_1 \arrow[r, "{\mathbf{W}_{is}, \mathbf{W}_{ds}}"] & \mathbf{s}_1 \arrow[r, "\mathbf{W}_{sc}"] \arrow[d, no head] & \mathbf{c}_1 \\
 & \cdots \arrow[d] & \\
\mathbf{i}_r, \mathbf{d}_{r+1} \arrow[r, "{\mathbf{W}_{is}, \mathbf{W}_{ds}}"] & \mathbf{s}_{r+1} \arrow[r, "\mathbf{W}_{sc}"] & \mathbf{c}_{r+1}
\end{tikzcd}$$

(a) The RNN configuration.

$$\begin{tikzcd}
\mathbf{q}^\frown\vec{\mathbf{0}}_i^\frown\vec{\mathbf{0}}_d \arrow[r, "\mathbf{I}"] & \boxed{\text{LSTM}} \arrow[r, "\mathbf{I}"] \arrow[d, dashed] & \mathbf{s}_0 & \\
\mathbf{0}_q^\frown\vec{\mathbf{0}}_i^\frown\vec{\mathbf{d}}_1 \arrow[r, "\mathbf{I}"] & \boxed{\text{LSTM}} \arrow[r, "\mathbf{I}"] \arrow[d, dashed, no head] & \mathbf{s}_1 \arrow[r, "\mathbf{W}_{sc}"] & \mathbf{c}_1 \\
 & \cdots \arrow[d] & & \\
\mathbf{0}_q^\frown\mathbf{i}_r^\frown\mathbf{d}_{r+1} \arrow[r, "\mathbf{I}"] & \boxed{\text{LSTM}} \arrow[r, "\mathbf{I}"] & \mathbf{s}_{r+1} \arrow[r, "\mathbf{W}_{sc}"] & \mathbf{c}_{r+1}
\end{tikzcd}$$

(b) The LSTM configuration.

Figure 4.2: Two possible implementations of the mappings $\mathcal{I}(\cdot), \mathcal{U}(\cdot)$ and the function $\mathcal{F}(\cdot)$. We use $\mathbf{q}$ to denote the vector representation of the query q, $\mathbf{i}_r$ to denote the vector representation of the interactions i_r, $\mathbf{d}_r$ to denote the vector representation of the document d_r, $\mathbf{s}_r$ to denote the vector state after examining the document d_r and $\mathbf{c}_r$ to denote the vector of size 1, whose component equals $P(C_r = 1 \mid q, i_1, \ldots, i_{r-1}, d_1, \ldots, d_r)$. We use the symbol $^\frown$ to denote the concatenation of two vectors and write $\vec{\mathbf{0}}_q$, $\vec{\mathbf{0}}_d$, $\vec{\mathbf{0}}_r$ to denote zero vectors of the same size as the query, document and interactions representations, respectively. The matrices $\mathbf{W}_{qs}, \mathbf{W}_{ss}, \mathbf{W}_{is}, \mathbf{W}_{ds}$ denote the projections applied to the vectors $\mathbf{q}$, $\mathbf{s}_r$, $\mathbf{i}_r$, $\mathbf{d}_{r+1}$; the matrix $\mathbf{I}$ denotes an identity matrix. The rectangles labeled LSTM denote the *long short-term memory* block [66] that is used to alleviate the *vanishing and exploding gradient* problem [9]. The matrix $\mathbf{W}_{sc}$ denotes the projection matrix from the vector state $\mathbf{s}_{r+1}$ to the vector $\mathbf{c}_{r+1}$. After each projection a non-linear transformation is applied. The LSTM block contains non-linear transformations inside.

information from the vector state $\mathbf{s}_r$ to the vector state $\mathbf{s}_{r+1}$:

$$\begin{aligned}\mathbf{s}_0 &= g_1(\mathbf{W}_{qs}\mathbf{q} + \mathbf{b}_1),\\ \mathbf{s}_{r+1} &= g_2(\mathbf{W}_{ss}\mathbf{s}_r + \mathbf{W}_{is}\mathbf{i}_r + \mathbf{W}_{ds}\mathbf{d}_{r+1} + \mathbf{b}_2).\end{aligned}$$

The functions $g_1(\cdot)$ and $g_2(\cdot)$ denote element-wise non-linear transformations.[5] The matrices $\mathbf{W}_{qs}, \mathbf{W}_{ss}, \mathbf{W}_{is}, \mathbf{W}_{ds}$ and the bias vectors $\mathbf{b}_1, \mathbf{b}_2$ are the parameters of $\mathcal{I}(\cdot), \mathcal{U}(\cdot)$, which are to be learned during training.

The probability $\mathbf{c}_{r+1}$ is computed using a fully-connected layer with one output unit and the sigmoid activation function:

$$\mathbf{c}_{r+1} = \sigma(\mathbf{W}_{sc}\mathbf{s}_{r+1} + \mathbf{b}_3). \tag{4.4}$$

The matrix $\mathbf{W}_{sc}$ and the bias vectors $\mathbf{b}_3$ are the parameters of $\mathcal{F}(\cdot)$, which are to be learned during training. The sigmoid function $\sigma(x)$ is used to ensure that the output falls in the interval (0, 1).

LSTM configuration

A possible problem of the RNN configuration is the *vanishing and exploding gradient* problem described by Bengio et al. [9]: after applying a few non-linear transformations, the norm of the gradient gets either too small (the vanishing gradient problem, where no learning is happening) or too large (the exploding gradient problem, where the values of the network parameters become unstable). To account for this problem, Hochreiter and Schmidhuber [66] propose the *long short-term memory* (LSTM) block. The LSTM block contains a memory cell (i.e., a vector) and three gate units (i.e., vectors of the same size as the memory cell): the *input gate* that filters out irrelevant information in the input vector, the *forget gate* that filters out information in the vector state that is no longer needed and the *output gate* that controls information in the output vector. It has been shown that the gate mechanism helps to alleviate the vanishing and exploding gradient problems; this yields better results than simple recurrent neural networks that do not use it [55, 66, 83].

The LSTM configuration is illustrated in Figure 4.2(b). Unlike the RNN configuration, which propagates the information from the vector state $\mathbf{s}_r$ to the vector state $\mathbf{s}_{r+1}$ directly, the LSTM configuration propagates it through the LSTM block, which, as said, helps to mitigate the vanishing and exploding gradient problem. The click probability $\mathbf{c}_r$ is computed as in the RNN configuration (Eq. 4.4). The parameters of the LSTM configuration, i.e., the parameters of the LSTM block and the parameters of the function $\mathcal{F}(\cdot)$, are learned during training.

Training RNN and LSTM configurations

Similar to PGM-based click models, both RNN and LSTM configurations are trained by maximizing the likelihood of observed click events. In particular, we optimize the logarithm of the likelihood function using the *stochastic gradient descent* (SGD) algorithm with mini-batches. The learning rates for each parameter are adjusted according to the

[5]The possible choices are the sigmoid function, the hyperbolic tangent and the rectified linear unit [8].

ADADELTA algorithm [177] (we use the default values of $\epsilon = 10^{-6}$ and $\rho = 0.95$). We also use the *gradient clipping* technique [130] to alleviate the exploding gradient problem [9] (we set the value of the threshold $= 1$).

We do not provide the expressions for computing the gradients of the logarithm of the likelihood function with respect to the configurations' parameters, because such expressions can be computed automatically using symbolic differentiation in math packages such as Theano [12].

The main message to take away from this section is that we use distributed representations (sequences of vector states as detailed in §4.3.1) to model user browsing behavior. We use neural click models to learn those representations. We write $\mathrm{NCM}_{\mathrm{X}}^{\mathrm{Y}}$ to denote a neural click model with representation X (QD, QD+Q, QD+Q+D) and configuration Y (RNN, LSTM). The neural click models can be used to simulate user behavior on a SERP and to infer document relevance from historical user interactions. We estimate the relevance of a document d to a query q using the probability of click on d when d appears on the first position, i.e., $P(C_1 = 1 \mid q, d)$.

4.4 Experimental Setup

In this section we describe our experimental setup. The research questions are outlined in §4.4.1. The dataset and baselines are described in §4.4.2 and §4.4.4. The evaluation methodology is given in §4.4.3. The experiments that we conduct to answer our research questions are outlined in §4.4.5.

4.4.1 Research questions

We split RQ 3 into six subquestions:

RQ 3.1 Does the distributed representation-based approach that models user behavior as a sequence of distributed vector representations have better predictive abilities than the PGM-based approach that models user behavior as a sequence of observed and hidden events?

RQ 3.2 Does the LSTM configuration have better learning abilities than the RNN configuration?

RQ 3.3 Does the representation $\mathbf{q}^{(2)}$ of a query q as defined in §4.3.2 provide the means to transfer behavioral information from historical query sessions generated by the query q to new query sessions generated by the query q?

RQ 3.4 Does the representation $\mathbf{d}^{(3)}$ of a document d as defined in §4.3.2 provide the means to transfer behavioral information from historical query sessions whose SERPs contain the document d, to new query sessions whose SERP contain the document d?

RQ 3.5 Do the neural click models produce better document rankings than the PGM-based models?

RQ 3.6 Do the neural click models operate with concepts similar to the ones used in the PGM-based models? In particular,

(a) Do they learn to account for the document rank similarly to most PGM-based click models?

(b) Do they learn to account for the distance to the previous click similarly to the state-of-the-art UBM model?

4.4.2 Dataset

We use the Yandex Relevance Prediction dataset,[6] which contains $146{,}278{,}823$ query sessions sampled from logs of Yandex, a major commercial search engine in Russia. There are a total of $30{,}717{,}251$ unique queries and $117{,}093{,}258$ unique documents in this dataset. The query sessions are ordered by time. We split the query sessions into two equal parts, and use the earlier query sessions to train click models and the later query sessions to evaluate their prediction performance.

The Yandex Relevance Prediction dataset also contains human-generated binary relevance labels for $41{,}275$ query-document pairs (4991 queries; on average, each query is associated with 8 documents). We use them to evaluate the ranking performance of our click models.

4.4.3 Evaluation methodology

We consider the click prediction task (i.e., predicting user clicks on search engine results) and the relevance prediction task (i.e., ranking documents by their relevance to a query).

Click prediction

The task is to predict user clicks on a SERP. The training and test sets consist of query sessions separated by time: the training set comprises query sessions from an earlier period; the test set comprises query sessions from a later period.

Following Dupret and Piwowarski [46], we evaluate the quality of click prediction using the *perpexity* metric, which measures how "surprised" the model is upon observing a particular set of clicks. In particular, we compute perplexity of a model M on a set of sessions S separately for each rank r as follows:

$$p_r(M) \quad = \quad 2^{-\frac{1}{|S|}\sum_{s\in S}\log_2 P_M\left(C_r=c_r^{(s)}\right)},$$

where $P_M(C_r = c_r^{(s)})$ denotes the probability of observing a click event $c_r^{(s)}$ (i.e., click or skip) at rank r in a query session s, as predicted by the click model M. The total perplexity of the click model M is calculated by averaging perplexities over all positions. Lower values of perplexity correspond to higher quality of a model.

[6]`http://imat-relpred.yandex.ru/en/datasets` (last visited August 16, 2018).

We also report the logarithm of the likelihood function $\mathcal{L}(M)$ for each click model M, averaged over all query sessions S in the test set (all click models are learned to optimize the likelihood function):

$$\log \mathcal{L}(M) \quad = \quad \frac{1}{|S|} \sum_{s \in S} \log P_M \left(C_1 = c_1^{(s)}, \ldots, C_n = c_n^{(s)} \right),$$

where $P_M(C_1 = c_1^{(s)}, \ldots, C_n = c_n^{(s)})$ denotes the probability of observing a particular sequence of clicks $c_1^{(s)}, \ldots, c_n^{(s)}$ in a query session s according to the click model M. We refer to this statistic as *log-likelihood*.

We perform significance testing using a paired t-test on per query session scores; the differences are considered statistically significant for p-values lower than 0.05.

Relevance prediction

Here, the task is to rank documents by their estimated relevance to a query. The training set consists of all query sessions, while the test set consists of query-document pairs that occur at least once in the training set and that have a human generated relevance label. We use the training set to train click models, and the test set to evaluate the quality of document rankings produced by click models.

Following Chapelle and Zhang [27], we evaluate the quality of document rankings using the mean *normalized discounted cumulative gain* (NDCG) [77]. We report NDCG scores at truncation levels 1, 3, 5 and 10.

We perform significance testing using a paired t-test on per query scores; the differences are considered statistically significant for p-values lower than 0.05.

4.4.4 Baselines

We use DBN, DCM, CCM and UBM as baseline click models.[7] Following [34], we set the priors of all click model parameters to a Beta distribution with $\alpha = 1$ and $\beta = 2$, and the number of EM iterations to 50.

The baseline performance on the click prediction task and the relevance prediction task is given in Tables 4.1 and 4.2. Table 4.1 shows that UBM is the best for click prediction (in terms perplexity and log-likelihood); Table 4.2 shows that CCM is the best for ranking (in terms of NDCG scores). These results agree with earlier work [57].

4.4.5 Experiments

We design our experiments to answer our research questions.

Experiment 1. To answer RQ 3.1, we compare the performance of $\text{NCM}_{\text{QD}}^{\text{RNN}}$ against UBM on the click prediction task (UBM is the best performing baseline click model on this task).

[7] We use the PyClick implementation available at `https://github.com/markovi/PyClick` (last visited August 16, 2018).

Table 4.1: Performance of the baseline click models on the click prediction task. Differences between all pairs of click models are statistically significant ($p < 0.001$). The best results are given in bold.

Click model	Perplexity	Log-likelihood
DBN	1.3510	-0.2824
DCM	1.3627	-0.3613
CCM	1.3692	-0.3560
UBM	$\mathbf{1.3431}$	$-\mathbf{0.2646}$

Table 4.2: Performance of the baseline click models on the relevance prediction task. Improvements of DCM and CCM over DBN and UBM are statistically significant ($p < 0.001$). Differences between (i) DBN and UBM, (ii) DCM and CCM are not statistically significant ($p > 0.05$). The best results are given in bold.

	NDCG			
Click model	@1	@3	@5	@10
DBN	0.717	0.725	0.764	0.833
DCM	0.736	0.746	0.780	0.844
CCM	$\mathbf{0.741}$	$\mathbf{0.752}$	$\mathbf{0.785}$	$\mathbf{0.846}$
UBM	0.724	0.737	0.773	0.838

Experiment 2. To answer RQ 3.2, we compare the performance of $\mathrm{NCM}^{\mathrm{RNN}}_{\mathrm{QD}}$ against $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q+D}}$ on the click prediction task; the best configuration (RNN or LSTM) is then denoted with B_2.

Experiment 3. To answer RQ 3.3, we compare the performance of $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD}}$, and $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD+Q}}$ on the click prediction task.

Experiment 4. To answer RQ 3.4, we compare the performance of $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD+Q}}$, and $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD+Q+D}}$ on the click prediction task.

Experiment 5. To answer RQ 3.5, we compare the performance $\mathrm{NCM}^{\mathrm{RNN}}_{\mathrm{QD}}$, $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD+Q}}$ and $\mathrm{NCM}^{\mathrm{B}_2}_{\mathrm{QD+Q+D}}$ against CCM on the relevance prediction task (CCM is the best performing baseline click model on this task).

Experiment 6. To answer RQ 3.6, we analyze vector states $\mathbf{s}_r$ with respect to document ranks and distances to the previous click. The analysis is performed by visualizing the vector states $\mathbf{s}_r$ of the best performing neural click model on the click prediction task in a two-dimensional space using the *t-SNE* dimensionality reduction technique [152].[8] We compute the vector states $\mathbf{s}_r$ on a uniformly sampled subset of the test query sessions.

In all experiments, we use vector states of size 256. The neural click models are trained using mini-batches of 64 query sessions and the parameters specified in §4.3.3.

[8] t-SNE is a state of the art method for visualizing high dimensional data that preserves distances between the data points.

4.5 Results

We present the outcomes of the experiments described in §4.4.5 and provide answers to our research questions stated in §4.4.1.

4.5.1 Click prediction task

The results for the click prediction task are given in Table 4.3 and Figures 4.3, 4.4. Table 4.3 lists the overall performance of the click models considered in terms of perplexity and log-likelihood. Figure 4.3 plots perplexity vs. the number of times a query occurred in the training set. Figure 4.4 shows perplexity values at different ranks.

Table 4.3: Performance on the click prediction task. Differences between all pairs of click models are statistically significant ($p < 0.001$). The best results are given in bold.

Click model	Perplexity	Log-likelihood
UBM	1.3431	-0.2646
$\text{NCM}^{\text{RNN}}_{\text{QD}}$	1.3379	-0.2564
$\text{NCM}^{\text{LSTM}}_{\text{QD}}$	1.3362	-0.2547
$\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$	1.3355	-0.2545
$\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$	$\mathbf{1.3318}$	$\mathbf{-0.2526}$

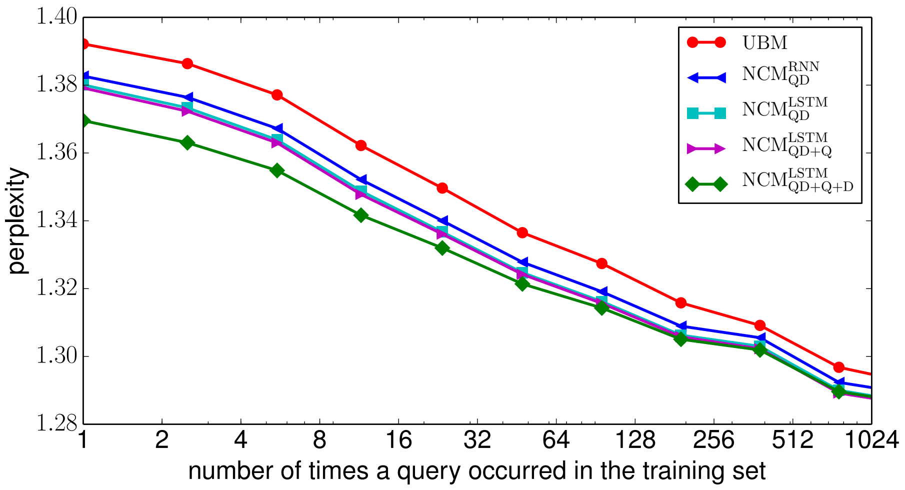


Figure 4.3: Perplexity for different query frequencies. (Best viewed in color.)

RQ 3.1. Table 4.3 shows that $\text{NCM}^{\text{RNN}}_{\text{QD}}$ outperforms the best performing PGM-based click model, UBM, in terms of perplexity and log-likelihood by a large margin. Figure 4.3 shows that $\text{NCM}^{\text{RNN}}_{\text{QD}}$ has lower perplexity than UBM for all query frequencies.

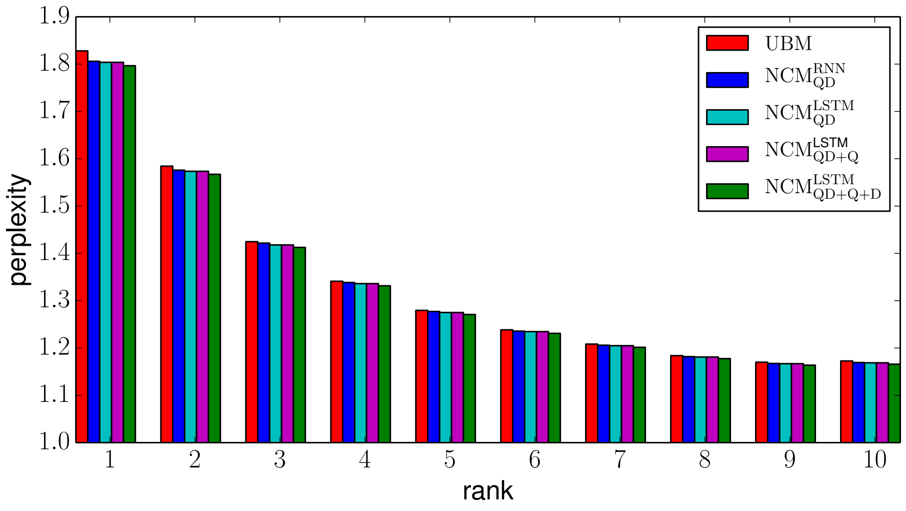


Figure 4.4: Perplexity for different ranks. (Best viewed in color.)

Figure 4.4 also shows that $\mathrm{NCM}^{\mathrm{RNN}}_{\mathrm{QD}}$ performs better than or, at least, as good as UBM at all ranks.

From the above results we conclude that the DR-based approach, which models user behavior as a sequence of distributed vector representations and learns patterns of user behavior directly from data, has better predictive abilities than the PGM-based approach used in state-of-the-art click models DBN, DCM, CCM and UBM, which models user behavior as a sequence of observed and hidden events and which requires a carefully hand-crafted set of rules describing user behavior (i.e., a probabilistic graphical model).

RQ 3.2. Table 4.3 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ outperforms $\mathrm{NCM}^{\mathrm{RNN}}_{\mathrm{QD}}$ in terms of perplexity and log-likelihood. $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ also performs better than, or at least as good as, $\mathrm{NCM}^{\mathrm{RNN}}_{\mathrm{QD}}$ for all query frequencies (Figure 4.3) and at all ranks (Figure 4.4).

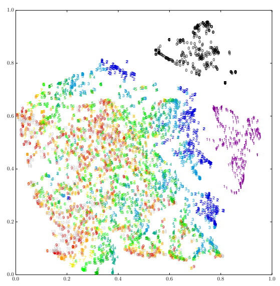
From the above results, we conclude that the introduction of the LSTM block helps to improve the learning abilities of the neural click models. Therefore, we use the LSTM configuration in the subsequent experiments.

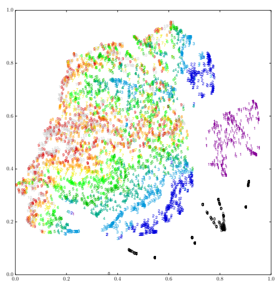
RQ 3.3. Table 4.3 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q}}$ outperforms $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ in terms of perplexity and log-likelihood by a small but statistically significant margin ($p < 0.001$). Figure 4.3 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q}}$ consistently outperforms $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ in terms of perplexity for all queries, with larger improvements observed for less frequent queries. In addition, Figure 4.4 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q}}$ performs as good as $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD}}$ in terms of perplexity at all ranks.

From the above results, we conclude that the representation $\mathbf{q}^{(2)}$ of a query q provides the means to transfer behavioral information between query sessions generated by the query q. And this, in turn, helps to better explain user clicks on a SERP.

RQ 3.4. Table 4.3 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q+D}}$ outperforms $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q}}$ in terms of perplexity and log-likelihood. Furthermore, Figure 4.3 shows that $\mathrm{NCM}^{\mathrm{LSTM}}_{\mathrm{QD+Q+D}}$ consistently outperforms $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ in terms of perplexity for rare and torso queries, with larger improvements observed for less frequent queries. Finally, Figure 4.4 shows that $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ outperforms $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ in terms of perplexity at all ranks.

From the above results, we conclude that the representation $\mathbf{d}^{(3)}$ of a document d provides the means to transfer behavioral information between query sessions, whose SERPs contain the document d. And this, in turn, helps to better explain user clicks on a SERP.


(a) All query sessions.


(b) Query sessions generated by queries that occur one time in the training set.

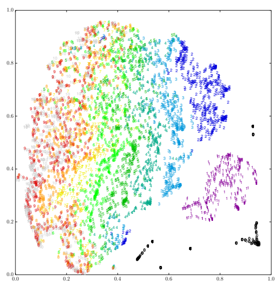

(c) Query sessions with no clicks generated by queries that occur one time in the training set.

Figure 4.5: Two-dimensional t-SNE projections of vector states $\mathbf{s}_r$ for different ranks r. Colors correspond to ranks: black 0; purple 1; dark blue 2; light blue 3; light blue-green 4; green 5; light green 6; yellow 7; orange 8; red 9; grey 10. (Best viewed in color.)

4.5.2 Relevance prediction task

The results for the relevance prediction task are given in Table 4.4, which lists the performance of the click models we consider in terms of NDCG scores at truncation levels 1, 3, 5 and 10.

Table 4.4: Performance on the relevance prediction task. Improvements of (i) $\text{NCM}^{\text{RNN}}_{\text{QD}}$, $\text{NCM}^{\text{LSTM}}_{\text{QD}}$ and $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ over CCM and (ii) $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ over the other neural click models are statistically significant ($p < 0.05$). The best results are given in bold.

	NDCG			
Click model	@1	@3	@5	@10
CCM	0.741	0.752	0.785	0.846
$\text{NCM}^{\text{RNN}}_{\text{QD}}$	0.762	0.759	0.791	0.851
$\text{NCM}^{\text{LSTM}}_{\text{QD}}$	0.756	0.759	0.789	0.850
$\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$	**0.775**	**0.773**	**0.799**	**0.857**
$\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$	0.755	0.755	0.787	0.847

RQ 3.5. Table 4.4 shows that all neural click models outperform the best performing PGM-based click model, CCM, in terms of NDCG. The differences between $\text{NCM}^{\text{RNN}}_{\text{QD}}$ and $\text{NCM}^{\text{LSTM}}_{\text{QD}}$ are not statistically significant. $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ outperforms $\text{NCM}^{\text{LSTM}}_{\text{QD}}$ and $\text{NCM}^{\text{LSTM}}_{\text{QD}}$ by a large margin. Interestingly, the performance of $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ falls behind that of $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$.

This shows that the neural click models produce better document rankings than the PGM-based models. The differences between the neural click models can be explained as follows. When ranking a query-document pair (q, d), $\text{NCM}^{\text{LSTM}}_{\text{QD}}$ uses behavior information from historical query sessions generated by the query q and whose SERPs contain the document d. $\text{NCM}^{\text{LSTM}}_{\text{QD+Q}}$ also uses behavioral information from all historical query sessions generated by the query q, which helps, e.g., to distinguish highly personalized SERPs and to discount observed clicks in these sessions. $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ also uses behavioral information from all historical query sessions, whose SERP contain the document d. However, this global information does not tell us much about the relevance of the document d to the query q. It does, though, inform us about the attractiveness of the document d, which leads to improvements on the click prediction task (see Experiment 4).

4.5.3 Concepts learned by NCM

The results of the analysis of the $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ vector states are given in Figures 4.5 and 4.6. Figure 4.5 shows the t-SNE projections of the vector states $\mathbf{s}_r$ for different ranks r. It contains Figures 4.5(a), 4.5(b) and 4.5(c), in which the vector states $\mathbf{s}_r$ are generated for different sets of query sessions: Figure 4.5(a) uses a uniformly sampled subset of query sessions in the test set (S_a); Figure 4.5(b) uses the query sessions in S_a that are generated by queries that occur one time in the training set (S_b); Figure 4.5(c) uses the query sessions in S_b that contain no clicks (S_c). Figure 4.6 plots the t-SNE projections of the vector state $\mathbf{s}_7$ for different distances to the previous click.[9] Here, we compute the vector states for the query sessions in S_a, and then filter out some vector states to construct a balanced set that contains equal number of vector states for each distance $d = 0, 1, \dots, 6$.

RQ 3.6 (a). Figure 4.5(a) shows how the vector states $\mathbf{s}_r$ for different ranks r are positioned in the space learned by $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$. We find that the subspaces of $\mathbf{s}_0$ and $\mathbf{s}_1$ are well separated from the subspaces of $\mathbf{s}_r$ computed at lower positions; the subspaces of $\mathbf{s}_2$ and $\mathbf{s}_3$ are also separated from the subspaces of $\mathbf{s}_r$ computed for other ranks, but have a significant overlap with each other. The subspaces of vector states $\mathbf{s}_r$ for ranks $r > 3$ have large overlaps with each other. We hypothesize that this is due to the fact that other factors (e.g., query frequency and clicks on previous documents) are becoming more important for modeling user behavior as rank r increases.

We test our hypothesis by fixing some of these factors. Figure 4.5(b) shows that for query sessions generated by queries of similar frequencies (in our case, by queries that occur one time in the training set), the subspaces of the vector states $\mathbf{s}_0, \dots, \mathbf{s}_6$ are much better separated than the subspaces of the vector states $\mathbf{s}_0, \dots, \mathbf{s}_6$ computed

[9]We choose $\mathbf{s}_7$ because the rank 7 is low enough to see a sufficient number of distances but not too low to suffer from sparsity.

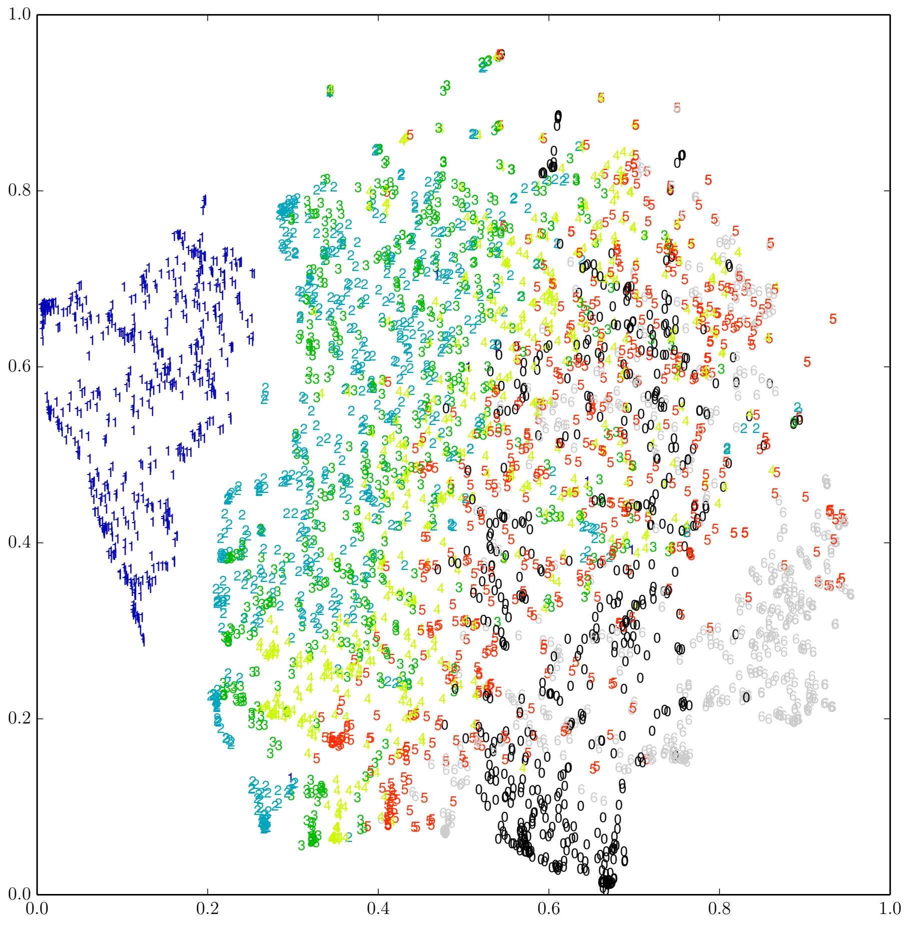


Figure 4.6: Two-dimensional t-SNE projections of the vector state $\mathbf{s}_7$ for different distances d to the previous click. Colors correspond to distances: black 0; blue 1; blue-green 2; green 3; green-yellow 4; red 5; grey 6. (Best viewed in color.)

for query sessions generated by all queries (Figure 4.5(a)). Furthermore, Figure 4.5(c) shows that for query sessions generated by queries of similar frequencies and having the same click pattern (in our case, no clicks) the subspaces of $\mathbf{s}_r$ are even better separated by ranks. This is intuitive, because the less information there is to explain user behavior (each query occurred only once and no clicks were observed), the more $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ learns to rely on ranks.

Interestingly, Figure 4.5(b) shows that the subspaces of the vector states $\mathbf{s}_r$ for $r > 1$ consist of more than one dense clusters (see, e.g., $\mathbf{s}_2$). We explain this by the fact that other factors, such as clicks on previous documents, are also memorized by $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$. In particular, Figure 4.5(c) shows that for query sessions generated by queries of the same frequency and having the same click pattern, the subspaces of the vector states consist of single dense clusters.

From the above results, we conclude that $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ learns the concept "current document rank" although we do not explicitly provide this concept in the document representation. In particular, $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ strongly relies on the current document rank to explain user browsing behavior on top positions. To explain user browsing behavior at lower positions, $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ considers other factors to be more important. However, $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ still discriminates most other ranks (we find this by limiting the set of query sessions, which are used to compute the vector states $\mathbf{s}_r$, to query sessions generated by queries of similar frequencies and having a particular set of clicks).

RQ 3.6 (b). Figure 4.6 shows how the vector states $\mathbf{s}_7$ for different distances to the previous click are positioned in the vector state space learned by $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$. Here, the distance to the previous click varies from 0 (no clicks above rank 7) to 6 (a click on a document at rank 6). We find that the subspace of the vector states $\mathbf{s}_7$ in query sessions containing a click at rank 6 (which corresponds to distance $d = 1$) is well separated from the subspace of $\mathbf{s}_7$ in other query sessions. The subspace corresponding to query sessions containing a click on the first position ($d = 6$) is also well separated from the subspace corresponding to other query sessions. The subspaces of the vector states $\mathbf{s}_7$ for distances $d \in \{0, 2, \ldots, 5\}$ have a significant overlap with each other. Still, the subspace corresponding to $d = 2$ is well separated from the subspace corresponding to $d = 5$. Interestingly, the subspace corresponding to query sessions containing no clicks on the first six documents ($d = 0$) has a larger overlap with the subspace corresponding to query sessions containing a click on the second position ($d = 5$) than with the subspace corresponding to query sessions containing a click on the first position ($d = 6$).

These results show that $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ learns the concept of distance to the previous click, although this information is not explicitly provided in the document representation. In particular, the information about a click on the previous document is particularly important. $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ also memorizes whether a user clicked on the first document. $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ distinguishes well between the distances 2 and 5, but the subspaces of the vector states computed for the pairs of distances $(2, 3)$, $(3, 4)$, $(4, 5)$ have a large overlap.

Zooming out, what we have learned from the t-SNE analysis of the $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ vector states is that $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ learns to account (a) for the document rank, and (b) for the distance to the previous click—the two most important concepts used in UBM, the state-of-the-art PGM-based click model. However, these are not the only concepts learned by $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$. All t-SNE projections contain a large number of clusters (of different density and size) that group vector states by their similarities in the vector state space learned by $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$. The large clusters are easily interpretable (e.g., they group vector states by rank, distance to the previous click). Smaller clusters are less easily interpretable, but their existence indicates that $\text{NCM}^{\text{LSTM}}_{\text{QD+Q+D}}$ also operates with concepts that are not hard-coded in PGM-based click models.

4.6 Conclusions and Future Work

In this chapter, we answered the following research question:

RQ 3 How to design a neural network that would be able to learn patterns in user click behavior directly from logged interaction data?

We represented user browsing behavior as a sequence of vector states that describes the information need of a user and the information available to the user during search. In contrast to existing click models, which represent user browsing behavior as a sequence of predefined binary events, the proposed representation is much richer, and thereby enables us to capture more complex patterns of user browsing behavior than existing click models. The proposed approach allows us to learn patterns of user behavior directly from interaction data, circumventing the need for a predefined set of rules, which are a key ingredient of existing click models.

We evaluated our approach on the click prediction task (i.e., to predict user clicks on search engine results) and the relevance prediction task (i.e., to rank documents by relevance based on historical user interactions). Our experimental results show that the proposed approach (i) has better predictive performance in terms of perplexity and log-likelihood for all query frequencies and at all ranks than DBN and UBM; and (ii) produces better document rankings in terms of NDCG than DBN and UBM. We also showed how to go beyond the level of query-document pairs and incorporate information (i) about all query sessions generated by a given query and (ii) about all query sessions containing a given document. The first yields a small improvement on the click prediction task and a considerable improvement on the relevance prediction task. The second yields a large improvement on the click prediction task and a significant deterioration on the relevance prediction task. Finally, we provided an analysis of the best performing model on the click prediction task, which showed that (i) it learned concepts such as "the current document rank" and "the distance to the previous click," which are used in UBM; and that (ii) it also learned other concepts that cannot be designed manually.

As to future work, first, we want to extend our models to non-linear user browsing behaviors, e.g., examining the first three documents and then clicking on the second document. Second, we want to consider other types of (i) user action, e.g., clicking on a sponsor advertisement, zooming on a result (in mobile search), reformulating a query; (ii) query, e.g., audio queries (in voice search), image queries (in image search), foreign language queries (in cross-lingual search); (iii) document, e.g., image results (in image search); and (iv) interaction, e.g., mouse movements. Third, we want to extend the modeling scope from a search engine result page to a search session. Fourth, we want to incorporate information about the user, e.g., user profile, location, time, etc.

Another interesting future direction is to analyze the vector states that we used in this chapter. A sequence of vector states provides a rich representation of the information need of a user and its evolution during the search. The information encoded in such a state can be used to detect positive abandonment or to distinguish user frustration from exploration of search engine results.

4.7 Appendix

Similarly to the neural click model framework described in §4.3.1, traditional click models can be fully described by mappings $\mathcal{I}(\cdot)$, $\mathcal{U}(\cdot)$ and the function $\mathcal{F}(\cdot)$ (see Eqs. (4.1), (4.2) and (4.3)). Below, we specify these parameters for UBM and DBN.

User browsing model

The vector state $\mathbf{s}_r$ can be represented with a tuple of four integer values (q, d, r, r'), where the first component q denotes a query ID, the second component d denotes the ID of a currently examined document, the third component r denotes the rank of the currently examined document and the last component r' denotes the rank of the previously clicked document.

The mapping $\mathcal{I}(\cdot)$ initializes $\mathbf{s}_0$ by setting the first component to the ID of a user's query and the other components to zero. The mapping $\mathcal{U}(\cdot)$ updates the previous state $\mathbf{s}_r$ to the next state $\mathbf{s}_{r+1}$ as follows: the second component is set to the ID of the next document d_{r+1}, the third component is incremented by one, and the fourth component is set to the third component of $\mathbf{s}_r$, if the previous document d_r was clicked. The interaction variable i_r denotes a click event: $i_r = 1$ if d_r was clicked, and $i_r = 0$ otherwise.

The function $\mathcal{F}(\cdot)$ computes the probability that a user clicks on the currently examined document as a product of the examination probability $\gamma_{r,r-r'}$ and the attractiveness probability $\alpha_{q,d}$. Both $\gamma_{r,r-r'}$ and $\alpha_{q,d}$ are the parameters of the function $\mathcal{F}(\cdot)$.

Overall, UBM can be formalized as follows.

$$\begin{aligned}
\mathcal{I}(q) &= (\text{QUERY\_ID}(q), 0, 0, 0) \\
\mathcal{U}(\mathbf{s}_r, i_t, d_{r+1}) &= (\mathbf{s}_r[1], \text{DOC\_ID}(d_{r+1}), \mathbf{s}_r[3] + 1, h(\mathbf{s}_r, i_r)) \\
h(\mathbf{s}_r, i_r) &= \begin{cases} \mathbf{s}_r[3] & \text{if } \ i_r = 1 \\ \mathbf{s}_r[4] & \text{otherwise} \end{cases} \\
\mathcal{F}(\mathbf{s}_{r+1}) &= \gamma_{\mathbf{s}_{r+1}[3], \mathbf{s}_{r+1}[3] - \mathbf{s}_{r+1}[4]} \cdot \alpha_{\mathbf{s}_{r+1}[1], \mathbf{s}_{r+1}[2]}
\end{aligned}$$

Dynamic Bayesian network

The vector state $\mathbf{s}_r$ can be represented with a tuple of two integer and one floating-point values (q, d, ϵ), where the first component q denotes a query ID, the second component d denotes the ID of a currently examined document, and the third component ϵ denotes the probability of examining the current document.

The mapping $\mathcal{I}(\cdot)$ initializes $\mathbf{s}_0$ by setting the first component to the ID of a user's query, the second component to zero and the third component to one. The mapping $\mathcal{U}(\cdot)$ updates the previous state $\mathbf{s}_r$ to the next state $\mathbf{s}_{r+1}$ as follows: the second component is set to the ID of the next document d_{r+1}, and the third component is updated according the function $g(\mathbf{s}_r, i_r)$:

$$g(\mathbf{s}_r, i_r) = \begin{cases} (1 - \beta_{\mathbf{s}_r[0], \mathbf{s}_r[1]})\gamma & \text{if } \ i_r = 1 \\ \dfrac{(1 - \alpha_{\mathbf{s}_r[0], \mathbf{s}_r[1]})\mathbf{s}_r[3]\gamma}{1 - \alpha_{\mathbf{s}_r[0], \mathbf{s}_r[1]}\mathbf{s}_r[3]} & \text{otherwise} \end{cases}$$

where γ, $\beta_{q,d}$, and $\alpha_{q,d}$ are the parameters of the mapping $\mathcal{U}(\cdot)$.

The above formula can be interpreted as follows (see [34, Chapter 3] for more details). If a user clicks on the current document ($i_r = 1$), then according to DBN she continues examining other documents if she is not satisfied with the current one ($1 - \beta_{q,d}$) and explicitly decides to continue (γ). The user skips the current document ($i_r = 0$) with probability $(1 - \alpha_{q,d})\epsilon/(1 - \alpha_{q,d}\epsilon)$, i.e., the user examines the current document (ϵ), but is not attracted by it ($1 - \alpha_{q,d}$), normalized by the total probability of no click ($1 - \alpha_{q,d}\epsilon$). In the case of a skip, the user continues examining other documents with probability γ.

The function $\mathcal{F}(\cdot)$ computes the probability that a user clicks on the currently examined document as a product of the examination probability ϵ and attractiveness probability $\alpha_{q,d}$. Here, $\alpha_{q,d}$ is the parameter of $\mathcal{F}(\cdot)$, which is shared with the mapping $\mathcal{U}(\cdot)$.

Overall, DBN can be formalized as follows.

$$\begin{aligned}
\mathcal{I}(q) &= (\text{QUERY\_ID}(q), 0, 1) \\
\mathcal{U}(\mathbf{s}_r, i_t, d_{r+1}) &= (\mathbf{s}_r[1], \text{DOC\_ID}(d_{r+1}), g(\mathbf{s}_r, i_r)) \\
\mathcal{F}(\mathbf{s}_{r+1}) &= \mathbf{s}_{r+1}[3] \cdot \alpha_{\mathbf{s}_{r+1}[1], \mathbf{s}_{r+1}[2]}
\end{aligned}$$

5 A Click Sequence Model

In Chapters 2, 3 and 4, we focused on modeling and predicting single interaction events, namely clicks. In this chapter, we for the first time focus on modeling and predicting sequences of interaction events. And in particular, sequences of clicks. We seek to answer the following research question asked in §1.1:

RQ 4 What are the challenges in predicting sequences of clicks and how to solve them?

We formulate the problem of click sequence prediction and propose a click sequence model (CSM) that aims to predict the order in which a user will interact with search engine results. CSM is based on a neural network that follows the encoder-decoder architecture. The encoder computes contextual embeddings of the results. The decoder predicts the sequence of positions of the clicked results. It uses an attention mechanism to extract necessary information about the results at each timestep. We optimize the parameters of CSM by maximizing the likelihood of observed click sequences.

We test the effectiveness of CSM on three new tasks: (i) predicting click sequences, (ii) predicting the number of clicks, and (iii) predicting whether or not a user will interact with the results in the order these results are presented on a SERP. Also, we show that CSM achieves state-of-the-art results on a standard click prediction task, where the goal is to predict an unordered set of results a user will click on.

5.1 Introduction

Search engines play an important role in our everyday lives. One way to improve them is by getting a better understanding of user search behavior, such as clicks, dwell times, mouse movements, etc. So far, models of user behavior have focused on modeling and predicting single events, e.g., clicks [34] and mouse movements [73], and properties of these events, e.g., time between clicks [16]. In this work for the first time we focus on modeling and predicting sequences of information interaction events and, in particular, sequences of clicks.

Although people tend to make only one (or sometimes no) click on a search engine result page (SERP), multi-click query sessions constitute a significant part of search traffic. For example, about 23% of the query sessions in the Yandex relevance prediction

This chapter is based on Borisov, Wardenaar, Markov, and de Rijke [20].

challenge dataset contain multiple clicks (see §5.4.1 for details). It is commonly assumed that users traverse search results from top to bottom, which leads to the assumption that clicks are ordered by the position of search results. However, it was shown that in practice this assumption does not always hold, and that up to 27.9%–30.4% of multi-click sequences, depending on the dataset, are not ordered by position [157].

We aim to create tools that help us understand, model and predict sequences of clicks on search engine results, which is important because it provides an opportunity for improving the user search experience. For example, knowing that a user is likely to click on many results or that there are high chances that the user will interact with the results in an order other than the one in which the results are presented on a SERP can be used by a search engine to proactively show an advice or make a change in the ranking.

We propose a *click sequence model* (CSM) that predicts a probability distribution over click sequences. At the core of our model is a neural network with encoder-decoder architecture. We implement the encoder as a bidirectional recurrent neural network (bi-RNN) that goes over the search engine result page from top to bottom and from bottom to top and outputs contextual embeddings of the results. We implement the decoder as a RNN with an attention mechanism. The decoder is initialized with the final states of the forward and backward RNNs of the encoder. It is used to predict the sequence of positions of the clicked results. The whole network is trained by maximizing the likelihood of the observed click sequences.

We evaluate our proposed CSM using a publicly available click log and show that CSM provides a good means to generate a short list of K click sequences that contains the observed click sequence with a high probability. We present an analysis of the performance of CSM for query sessions with different numbers of clicks and query sessions in which clicks are ordered/not ordered by position. We measure the performance of CSM on three new tasks: predicting click sequences, predicting the number of clicks and predicting ordered/unordered sequences of clicks. Additionally, we show that CSM achieves state-of-the-art results on the standard click prediction task, which allows us to compare CSM to traditional click models that model and predict single events, namely clicks.

Overall, we make the following contributions:

C1 We formulate a novel problem of predicting click sequences.

C2 To solve this problem, we propose a click sequence model (CSM) based on neural networks.

C3 We evaluate CSM on a range of prediction tasks, namely predicting click sequences, predicting the number of clicks, predicting ordered/unordered sequences of clicks and, finally, predicting clicks themselves.

As to the potential impact of the proposed CSM model, we believe it can be used to predict that (i) a user will click on more than one result, which may indicate that a user has a complex information need [96]; or that (ii) a user will interact with the results not in the order in which these results are presented on the SERP, which may indicate that a user is struggling and there are problems in the ranking of the results [125]. CSM can help us identify queries for which there is room for improvement (in terms of user experience) and it can serve as a quick analysis tool to interpret how a particular change in the ranking of the results will influence user click behavior.

The rest of the chapter is structured as follows. In §5.2 we provide a precise statement of the click sequence modeling and prediction problems that we are tackling. §5.3 introduces our neural network based model for predicting click sequences. In §5.4 we describe the setup of our experiments and §5.5 presents the results of those experiments. We describe related work in §5.6 and conclude in §5.7.

5.2 Problem Statement

In this section, we formulate the problem of click sequence prediction (§5.2.1) and propose three prediction tasks that can be solved by a model capable of predicting click sequences (§5.2.2).

5.2.1 Problem

Since the number and order of clicks may vary even for the same query and ranking of results (e.g., due to different users and contexts), there exists no unique *correct* click sequence, but a (possibly infinite) set of *probably correct* sequences does exist. Therefore, our main goal is to build a model that, given a query and a ranking of results, describes these probably correct click sequences.

To achieve this goal, we define the *click sequence prediction problem* as follows. First, we learn a probability distribution $\mathcal{M}$ over all possible click sequences. Second, we use this learned distribution to obtain the K most probable click sequences. These K sequences are then used to reason about the properties of the set of probably correct sequences mentioned above, e.g., predicting the expected number of clicks, the expected order of clicks, etc.

More formally, we define a click sequence model $\mathcal{M}$ as follows:

$$\mathcal{M}: \quad P(s \mid q, r_1, \ldots, r_N), \tag{5.1}$$

where q is a query, $r_1, \ldots, r_N$ is an ordered list of results and s is a sequence of positions of the clicked results $(p_1, \ldots, p_S)$.

5.2.2 Prediction tasks

There are many possible applications for a model $\mathcal{M}$ that is capable of (i) predicting a probability distribution over click sequences (Eq. 5.1), and (ii) retrieving the K most probable click sequences. It can be used to simulate user behavior, which is important in *online learning to rank* research [68, 142], or as a tool for analyzing how a particular change in the ranking of results will influence user click behavior. However, we do not investigate these applications in this work. Instead, we address three tasks that are both practically useful and help to evaluate the performance of the model $\mathcal{M}$.

Task 1 (predicting the number of clicks)

The goal of this task is to predict on how many results a user will click. Clicking on more than one result might indicate that a user has a complex information need.

Clicking on more than three or four results might indicate that a user is struggling or doing an exhaustive search [63, 125]. Both signals can be used by a search system to proactively show an advice or make a change in the ranking. Thus, we formally define Task 1 as predicting whether a user will click on $\leq L$ results.

To estimate the probability of clicking on $\leq L$ results, we generate a large number K (e.g., $K = 1024$) most probable click sequences $s_1, \ldots, s_K$ and marginalize over those sequences that have $\leq L$ clicks:

$$P(|s| \leq L) = \sum_{s \in \{s_1, \ldots, s_K\}} P(s) \mathbb{1}[|s| \leq L]. \tag{5.2}$$

Task 2 (predicting non-consecutive click sequences)

The goal of this task is to predict whether a user will interact with results in the order these results are presented on a SERP or in a different order, which we refer to as a *non-consecutive* order. Interacting with results in a non-consecutive order might indicate that a user is struggling [140]. As mentioned in Task 1, such a signal can be used by a search engine to proactively show an advice or make a change in the ranking.

Similarly to Task 1, we estimate the probability of clicking on results in a non-consecutive order by summing probabilities of the K most probable click sequences $s_1, \ldots, s_K$ according to $\mathcal{M}$ in which a user clicks on a result r_i after clicking on a result r_j located below r_i ($i < j$):

$$P(\downarrow\uparrow) = \sum_{s \in \{s_1, \ldots, s_K\}} P(s) \mathbb{1}[s \text{ is non-consecutive}]. \tag{5.3}$$

Task 3 (predicting clicks)

The last task is actually a standard task solved by click models [34]. The goal is to predict a subset of the presented results $r_1, \ldots, r_N$ on which a user will click. Being able to predict that a user will not interact with a subset of results, opens the door for reranking [172].

Similarly to Task 1 and Task 2, we estimate the click probability for position p by summing probabilities of the K most probable click sequences $s_1, \ldots, s_K$ according to $\mathcal{M}$ in which a user clicks on that position:

$$P_{\text{click}}(p \mid q, r_1, \ldots, r_N) = \sum_{s \in \{s_1, \ldots, s_K\}} P(s) \mathbb{1}[p \in s]. \tag{5.4}$$

In practice, it is probably better to use simpler models to predict clicks. So we use this task mainly to compare with existing work. We expect the results for a good model $\mathcal{M}$ to be not much worse compared to the results for click models specifically developed for this task [34]. In fact, as we show in §5.5, CSM achieves state-of-the-art performance on this task.

5.3 Method

In this section we propose the *click sequence model* (CSM), a model for predicting click sequences. We use s to denote a sequence of positions of the clicked results with a special end of sequence (EOS) token appended to it, i.e., $s = (p_1, \dots, p_k, \mathrm{EOS})$.

CSM is a neural network that is trained to maximize the likelihood of observed click sequences:

$$\mathcal{L}(s_1, \dots, s_{|S|}) \to \max_{\Theta}, \tag{5.5}$$

where Θ denotes the network parameters and $S = (s_1, \dots, s_{|S|})$ denotes click sequences used for training.

The network consists of two parts, called *encoder* and *decoder*. The encoder takes a user's query q and a list of search engine results $r_1, \dots, r_N$ as input and computes embeddings of the results, $\mathbf{r}_1, \dots, \mathbf{r}_N$. The embedded results $\mathbf{r}_1, \dots, \mathbf{r}_N$ are passed to the decoder, which at each timestep $t = 0, 1, \dots$ outputs a probability distribution over $N + 1$ positions. Positions $1, \dots, N$ correspond to clicking on the results $r_1, \dots, r_N$. The $(N + 1)$-th position corresponds to predicting that there will be no clicks (EOS). Upon observing a click at timestep t, the decoder updates its current state using the position p_t of the clicked result. Figure 5.1 illustrates the workflow in the form of a UML sequence diagram.

In §5.3.1 we discuss the implementation of the encoder and decoder. Then, in §5.3.2 we explain how to achieve the main goal of this study, i.e., predict K most probable click sequences using CSM. Finally, in §5.3.3 we specify training details.

5.3.1 Network architecture

Encoder

The aim of the encoder is to obtain information from a user's query q and results $r_1, \dots, r_N$ presented on a SERP, and pass this information to the decoder. We represent the encoded information as a list of embeddings $\mathbf{q}, \mathbf{r}_1, \dots, \mathbf{r}_N$, where each result embedding $\mathbf{r}_i$ should contain: (i) information about the result r_i, (ii) the results surrounding r_i and (iii) the query q. Below we sometimes use $\mathbf{r}_0$ instead of $\mathbf{q}$ to simplify the notation.

We propose to implement the encoder as a bidirectional recurrent neural network, which goes over the SERP in the top down order, i.e., $q, r_1, \dots, r_N$, and in the reverse order, i.e., $r_N, \dots, r_1, q$. The first, *forward RNN*, produces embeddings $\overrightarrow{\mathbf{q}} (= \overrightarrow{\mathbf{r}_{0:0}}), \overrightarrow{\mathbf{r}_{0:1}}, \dots, \overrightarrow{\mathbf{r}_{0:N}}$. The second, *backward RNN*, produces embeddings $\overleftarrow{\mathbf{r}_{N:N}}$, $\dots$, $\overleftarrow{\mathbf{r}_{N:1}}$, $\overleftarrow{\mathbf{q}} (= \overleftarrow{\mathbf{r}_{N:0}})$. These embeddings are concatenated to form the final embeddings $\mathbf{q}(= \mathbf{r}_0), \mathbf{r}_1, \dots, \mathbf{r}_N$ produced by the encoder.

We represent $q, r_1, \dots, r_N$ using the best performing behavioral features proposed in Chapter 4. These features count the number of times a particular click pattern, i.e., a set of positions of the clicked results, was observed on a SERP. A query q is represented as a 2^N dimensional vector, where each component counts the number of times a click pattern was observed in query sessions generated by q. The representation of a search result r consists of two parts, both of size $N2^N$. The components of the first part count, for each position $p = 1, \dots, N$, the number of times a click pattern was observed

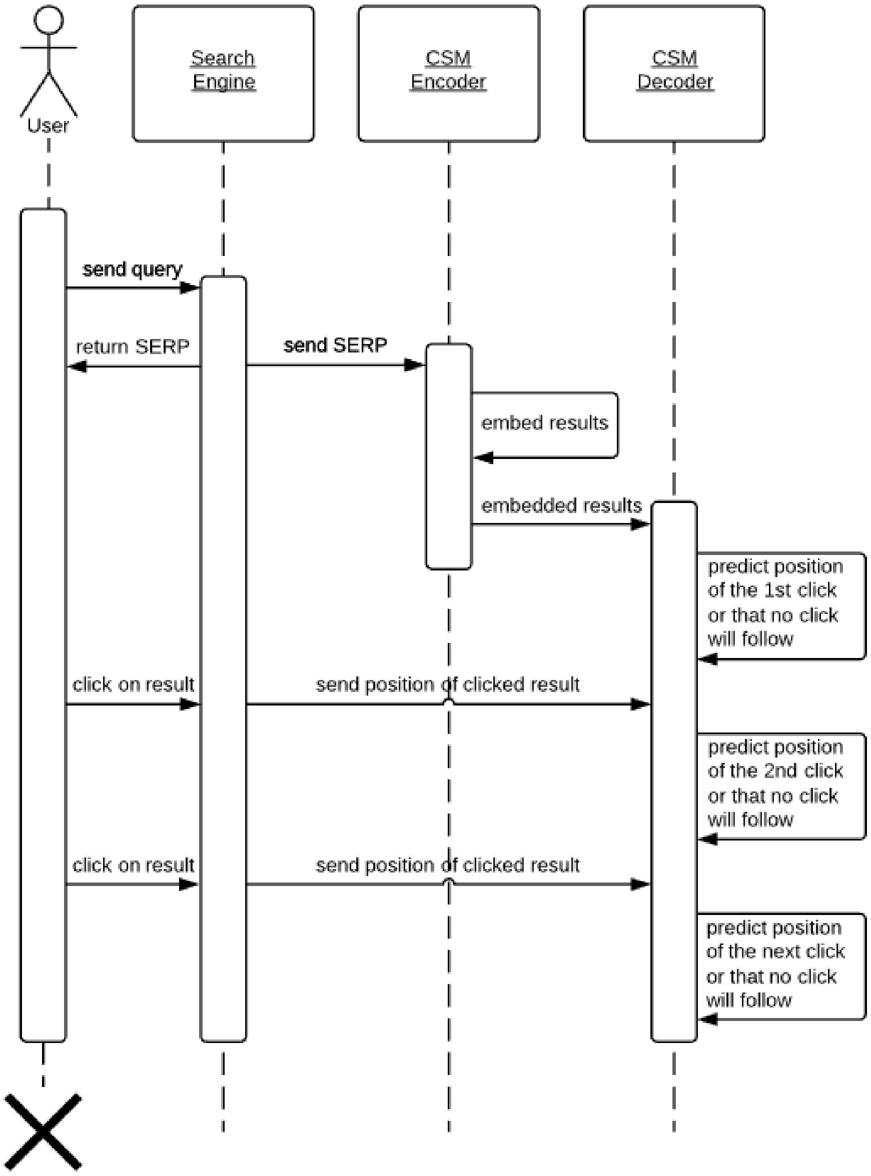

User
Search Engine
CSM Encoder
CSM Decoder
send query
return SERP
send SERP
embed results
embedded results
predict position of the 1st click or that no click will follow
click on result
send position of clicked result
predict position of the 2nd click or that no click will follow
click on result
send position of clicked result
predict position of the next click or that no click will follow

Figure 5.1: Modeling click sequences with CSM.

in query sessions in which r appears on position p. The components of the second part are similar, but include only query sessions generated by q. We apply a linear transformation to these sparse behavioral features to obtain the embeddings $\mathbf{x}_0, \ldots, \mathbf{x}_N$ of $q, r_1, \ldots, r_N$, which are passed to the RNNs.

We describe the encoder formally using Eqs. 5.6–5.9:

$$\mathbf{x}_i = \begin{cases} \text{Embed}(q) & i = 0 \\ \text{Embed}(r_i) & i = 1, \dots, N \end{cases} \tag{5.6}$$

$$\overrightarrow{\mathbf{r}_{0:0}}, \dots, \overrightarrow{\mathbf{r}_{0:N}} = \text{RNN}_{\text{forward}}(\mathbf{x}_0, \dots, \mathbf{x}_N) \tag{5.7}$$

$$\overleftarrow{\mathbf{r}_{N:N}}, \dots, \overleftarrow{\mathbf{r}_{N:0}} = \text{RNN}_{\text{backward}}(\mathbf{x}_0, \dots, \mathbf{x}_N) \tag{5.8}$$

$$\mathbf{r}_i = [\overrightarrow{\mathbf{r}_{0:i}}, \overleftarrow{\mathbf{r}_{N:i}}] \qquad (i = 0, \dots, N) \tag{5.9}$$

Decoder

The aim of the decoder is to predict a probability distribution over $(N+1)$ positions at each timestep. (As mentioned at the start of §5.3, the $(N+1)$-th position corresponds to predicting that there will be no clicks.) To make a good prediction at timestep $(t+1)$, we need to incorporate into the decoder the information about the position p_t of the result clicked at timestep t.

We propose to implement the decoder as an RNN that at each timestep $t = 0, 1, \dots$ outputs a vector $\mathbf{o}_t$ used to predict the probability distribution, and updates its hidden state using the position p_t of the observed click at timestep t. We also use an attention mechanism [5] to help the decoder extract the most relevant information from the list of embeddings $\mathbf{r}_0, \dots, \mathbf{r}_N$ at each timestep.

We initialize the hidden state of the decoder RNN using the concatenation of the final states of the forward and backward RNNs of the encoder, $[\overrightarrow{\mathbf{r}_{0:N}}, \overleftarrow{\mathbf{r}_{N:0}}]$, passed through a linear transformation; we use W_{init} to denote the transformation matrix. To obtain the probability distribution over $(N+1)$ positions at timestep t, we concatenate the vector $\mathbf{o}_t$ predicted by the decoder RNN and the attention vector $\mathbf{a}_t$ computed at timestep t, and pass the result through a linear transformation W_{output} followed by softmax.[1] We represent the position p_t of the observed click at timestep t as a one-hot vector $\mathbf{p}_t$ of size N. And we apply a linear transformation W_{pos} to it before passing to the decoder RNN.

We describe the decoder formally using Eqs. 5.10–5.13.

$$\mathbf{s}_0 = W_{\text{init}}[\overrightarrow{\mathbf{r}_{0:N}}, \overleftarrow{\mathbf{r}_{N:0}}] \tag{5.10}$$

$$\mathbf{a}_{t+1} = \text{Attention}(\mathbf{s}_t, [\mathbf{r}_0, \dots, \mathbf{r}_N]) \tag{5.11}$$

$$\mathbf{s}_{t+1}, \mathbf{o}_{t+1} = \text{RNN}_{\text{step}}(\mathbf{s}_t, \mathbf{a}_t, W_{pos}\mathbf{p}_t) \tag{5.12}$$

$$P(p_{t+1} \mid \dots) = \text{Softmax}\left(W_{\text{output}}[\mathbf{o}_{t+1}, \mathbf{a}_{t+1}]\right) \tag{5.13}$$

To alleviate the *exploding gradient problem* [9], we use gated recurrent units (GRUs) in both the forward and backward RNNs of the encoder, and in the decoder RNN.

5.3.2 Beam search

As stated in §5.2.1, our main goal is to predict K most probable click sequences for a given query and search results. These K sequences are then used to reason about actual

[1] Using the output of an RNN together with an attention vector has been shown to improve prediction performance [5, 129]. In the literature, this idea is known as *deep output* [129].

user click behavior, i.e., sequences of clicks that a user could actually perform for the given query and search results (we call them *probably correct* sequences, see §5.2.1).

CSM defines a probability distribution over infinitely many click sequences. Extracting the K most probable sequences in this case is not straightforward (since we cannot simply go over all sequences and pick the K best ones). We need a means of generating the K most probable sequences without having to calculate the probability for every possible click sequence. To do that, we suggest to use beam search [54].

In our experiments, we use $K \leq 1024$ and *beam size* $= K$. Setting the beam size to K guarantees that the K sequences generated by beam search have the highest probabilities according to CSM, i.e., they are indeed the most probable click sequences according to CSM. Using a smaller beam size allows us to generate K sequences faster, but does not guarantee that these sequences are the most probable ones.

5.3.3 Training

We learn the parameters Θ of the CSM network (both the encoder and decoder parts) by maximizing the log-likelihood of observed click sequences. We optimize these parameters using stochastic gradient descent (SGD). The learning rates for each parameter are adjusted according to the Adam [90] algorithm using the default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. We also use *gradient clipping* [130] with the norm set to 1 to alleviate the *exploding gradient problem* [9], which, as mentioned earlier, GRUs also try to mitigate.

5.4 Experimental Setup

In this section we describe our experimental setup. We start by describing the data we use to conduct our experiments (§5.4.1). Then we discuss our evaluation methodology (§5.4.2), formulate research questions (§5.4.3) and list the experiments we run to answer these research questions (§5.4.4).

5.4.1 Data

We use the Yandex Relevance Prediction dataset[2] released in 2011 by Yandex, the major search engine in Russia. The dataset consists of 146,278,823 query sessions ordered by time. We use the first half of the dataset for training CSM, and 100,000 randomly selected query sessions from the second half of the dataset for evaluation. In Chapter 4 we also use the first half of the dataset for training, which allows a direct comparison with the results reported in this chapter.

The statistics about the number of query sessions in the test set split by the number and order of clicks is given in Table 5.1.

[2] `https://academy.yandex.ru/events/data_analysis/relpred2011/` (last visited August 16, 2018)

Table 5.1: The number of query sessions in the test set split by the number and order of clicks. By ordered click sequences we mean those where a user clicks on results in the order they appear on a SERP. The total number of click sequences in the test set is 100,000.

Number of clicks	Ordered sequences	Unordered sequences
0	30,466	0
1	46,550	0
2	8851	2143
3	3437	1856
4	1564	1290
5	751	814
6	407	512
7	244	305
8	156	195
9	85	137
10+	73	164

5.4.2 Evaluation methodology

To properly evaluate the proposed CSM model, we would need to know a set of all (or at least a sample of) probably correct click sequences for each test query and search results. Then we could measure how well the K most probable sequences predicted by CSM describe the properties of the known probably correct sequences.

In practice, however, we observe only one (or a few) of all probably correct click sequences and, therefore, we cannot even argue about their properties. The best we can do is to check whether the observed click sequence appears in the list of K most probable sequences predicted by CSM. In particular, we measure recall@K, i.e., the fraction of query sessions for which CSM includes the observed click sequence in the list of K most probable sequences.

Since CSM is the first model for predicting click sequences, there are no baselines to compare against. However, we can use as a reference level the percentage of query sessions in which users interact with results in the order these results are presented on a SERP. In our test data, this percentage equals 92.73%. This is an upper bound under the assumption that a user scans search results sequentially from top to bottom. This means that a model, that predicts only click sequences ordered by position, will contain the observed click sequence in the list of K most probable sequences for $\leq 92.73\%$ of query sessions.

Task 1 (predicting the number of clicks)

Predicting whether a user will click on $\leq L$ results is a new task and, hence, there are no standard metrics to evaluate performance on this task and no existing baselines to compare to.

We propose to evaluate the performance on this task using perplexity and, because this is a classification problem for a fixed L, area under curve (AUC). Perplexity measures how "surprised" a model is upon observing $\leq L$ clicks. AUC measures the model's discriminative power.

We use a naive baseline which predicts that a user will make $\leq L$ clicks with a constant probability calculated on the training set. The AUC of such a method is 0.5.

Task 2 (predicting non-consecutive click sequences)

Predicting whether a click sequence will be ordered by position is also a new task and, similarly to Task 1, there are no standard metrics to evaluate the performance on this task and no existing baselines to compare to.

Similarly to Task 1, we use perplexity and AUC. Our naive baseline predicts that a click sequence will be ordered by position with a constant probability calculated on the training set. The AUC of such a method is also 0.5, as in Task 1.

Task 3 (predicting clicks)

Following [46, 57, 59, 157], we evaluate the performance on the standard click prediction task using perplexity, which measures how "surprised" a model is upon observing an unordered set of clicks on search engine results.

We use the following click models as our baselines: DBN [27], DCM [60], CCM [60], UBM [46] and NCM described in Chapter 4.

5.4.3 Research questions

We split RQ 4 into four subquestions:

RQ 4.1 How well does CSM, described in §5.3, predict probably correct click sequences?

- **(a)** For how many query sessions does the observed click sequence occur in the list of K most probable click sequences predicted by CSM? How fast does this number increase with K?
- **(b)** How well does CSM perform for query sessions in which clicks (i) follow the order in which results are presented on a SERP, and (ii) do not follow the order in which results are presented on a SERP.
- **(c)** How well does CSM perform for query sessions with different numbers of clicks?
- **(d)** Do the K most probable click sequences provide a good means to reason about the probability distribution over click sequences predicted by CSM?

RQ 4.2 How well does CSM predict the number of clicks on search results (see Task 1 in §5.2.2)?

RQ 4.3 How well does CSM predict whether or not a user will click on results in the order they are presented on a SERP (see Task 2 in §5.2.2)?

RQ 4.4 How well does CSM predict clicks on search results (see Task 3 in §5.2.2)? Does it reach the performance of the state-of-the-art click models?

5.4.4 Experiments

We design our experiments to answer our research questions.

Experiment 1 (a). To answer RQ 4.1 (a), we measure the percentage of query sessions for which the observed click sequence occurs in the list of K most probable click sequences according to CSM. We use $K = \{1, 2, 3, \dots, 1024\}$.

Experiment 1 (b). To answer RQ 4.1 (b), we measure the percentage of query sessions for which the observed click sequence occurs in the list of K most probable click sequences according to CSM separately (i) for query sessions in which clicks are ordered by position, and (ii) for query sessions in which clicks are not ordered by position. We use $K = \{1, 2, 3, \dots, 1024\}$.

Experiment 1 (c). To answer RQ 4.1 (c), we measure the percentage of query sessions for which the observed click sequence occurs in the list of K most probable click sequences according to CSM for query sessions with $\leq L$ clicks. We use $K = \{1, 2, 3, \dots, 1024\}$ and $L = \{1, 2, 3, 4, 5\}$.

Experiment 1 (d). To answer RQ 4.1 (d), we compute the total probability of the K most probable click sequences according to CSM. If this probability is close to 1, we conclude that using the K most probable click sequences is enough to form a representative empirical distribution over click sequences. And, thus, the K most probable click sequences provide a good means to reason about the properties of the probability distribution over click sequences predicted by CSM. If the total probability mass of the K most probable click sequences is small, we conclude that using these sequences is not enough to reason about the probability distribution over click sequences predicted by CSM. We use $K = \{1, 2, 3, \dots, 1024\}$.

Experiment 2. To answer RQ 4.2, we compute probabilities of clicking on $\leq L$ results by marginalizing over the K most probable click sequences according to CSM (see Eq. 5.2). We use these probabilities to compute perplexity and AUC. We use $K = 1024$ and $L = \{1, 2, 3, 4, 5\}$.

Experiment 3. To answer RQ 4.3, we compute the probability that a user will click on results in the order these results are presented on a SERP by marginalizing over the K most probable click sequences according to CSM (see Eq. 5.3). We use this probability to compute perplexity and AUC, $K = 1024$.

Experiment 4. To answer RQ 4.4, we compute probabilities of clicking on each result by marginalizing over the K most probable click sequences according to CSM (see Eq. 5.4). We use these probabilities to compute perplexities for each position and average these perplexity values over positions to obtain the final score. We use $K = 1024$.

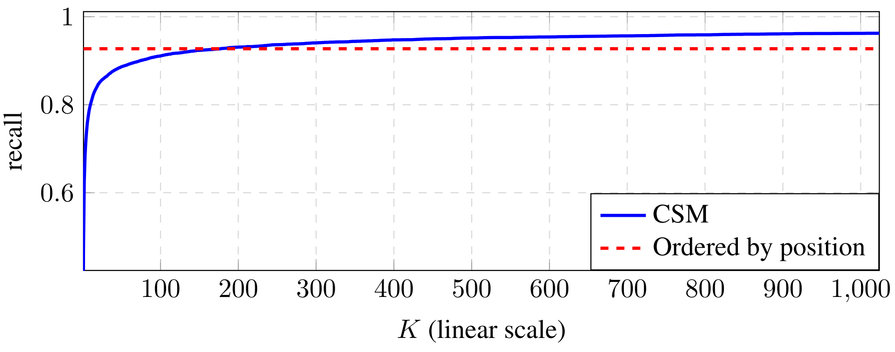
In our experiments, we use embeddings of size 256, and the same number of GRUs in all RNNs. We train CSM using stochastic gradient descent with mini-batches of 64 query sessions and the parameters specified in §5.3.3.

5.5 Results

In this section we present the results of the experiments described in §5.4.4 and provide answers to the research questions stated in §5.4.3.

5.5.1 Experiment 1 (a)

Figure 5.2 shows recall at different values of K (i.e., the percentage of query sessions for which the observed click sequence occurs in the list of K most probable sequences predicted by CSM) in linear and logarithmic scales. The percentage of query sessions in which clicks are ordered by position equals 92.73%. We show it on the plots as a reference level.



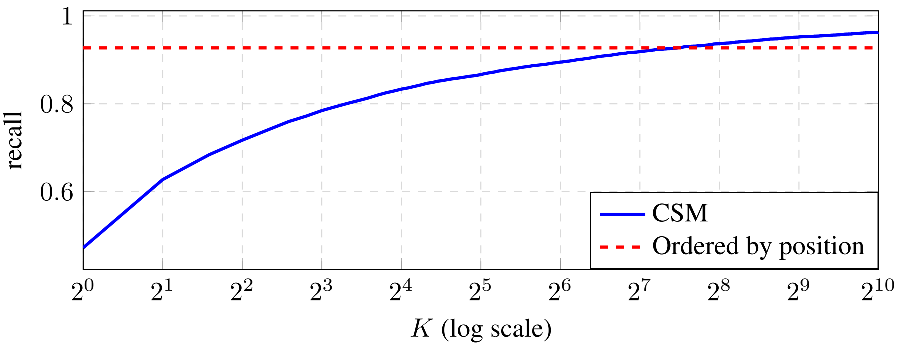


Figure 5.2: Percentage of query sessions for which the observed click sequence occurs in the list of K most probable click sequences predicted by CSM (blue, solid) and percentage of click sequences ordered by position (red, dashed).

We find that for 47.24% of query sessions, CSM assigns the highest probability to the observed click sequence. For 62.76% of query sessions, the observed sequence appears in the list of two sequences with the highest probabilities according to CSM. Since the curve on the logarithmic scale is concave (see Figure 5.2, bottom plot), we conclude that recall of CSM increases slower than logarithmically with K. The percentage of query sessions in which clicks are ordered by position can be seen as an upper bound under the assumption that a user scans search results sequentially from top to bottom. CSM does not make this assumption, and, as a result, is able to reach and surpass this upper bound, achieving 96.26% recall at $K = 1024$.

Answering RQ 4.1 (a), we conclude that recall of CSM increases slower than logarithmically with K, starting from 47.24% at $K = 1$ and reaching 96.26% at $K = 1024$, which is higher than recall under the sequential assumption (92.73%).

5.5.2 Experiment 1 (b)

Figure 5.3 shows recall at different values of K (in linear and logarithmic scales) for (i) all query sessions (black, solid), (ii) query sessions in which clicks are ordered by position (blue, solid), and (iii) query sessions in which clicks are not ordered by position (red, solid). Dashed lines show percentages of query sessions in the corresponding groups, in which clicks on results happen in the order these results are presented on a SERP. Obviously, the second group has 100% of query sessions with clicks ordered by position (and, hence, the blue dotted line denotes recall of 1), while the third group has no such sessions (and, hence, the red dotted line denotes recall of 0).

As we find in §5.5.1, CSM assigns the highest probability to the observed click sequence for 47.24% of query sessions. If we consider only query sessions in which clicks are ordered by position, this percentage goes up to 50.94% and then increases slower than logarithmically with K (see Figure 5.3, bottom plot) achieving 99.62% at $K = 1024$. However, if we consider only query sessions in which clicks are not ordered by position, this percentage goes down to 0 and then increases logarithmically with K for $K \geq 5$ (see Figure 5.3, bottom plot), achieving 53.37% at $K = 1024$.

It is to be expected that predicting click sequences, where clicks are not ordered by position, is a much more difficult task than predicting ordered clicks. First, in our training data the number of ordered click sequences is greater than the number of unordered click sequences (see Table 5.1 for the statistic computed on the test set; the training set shares the same distribution). Second, the number of possible ordered click sequences is less than the number of possible unordered click sequences. For click sequences of length L, there are $\binom{N}{L} = \frac{N!}{L!(N-L)!}$ possible ordered click sequences and N^L possible unordered click sequences.

And this is where CSM makes a difference: in more than 50% of cases, the observed click sequence appears in the top $K = 1024$ sequences predicted by CSM. Note that under the assumption that a user scans search results sequentially from top to bottom such sequences cannot be predicted at all (see the red dotted line at zero recall). Even in a simple case of predicting ordered click sequences, CSM almost reaches the perfect recall of 1 for $K = 1024$.

Answering RQ 4.1 (b), we conclude that recall of CSM is much higher in query sessions in which clicks follow the presentation order than in those in which users click on higher ranked results after clicking on a lower ranked result.

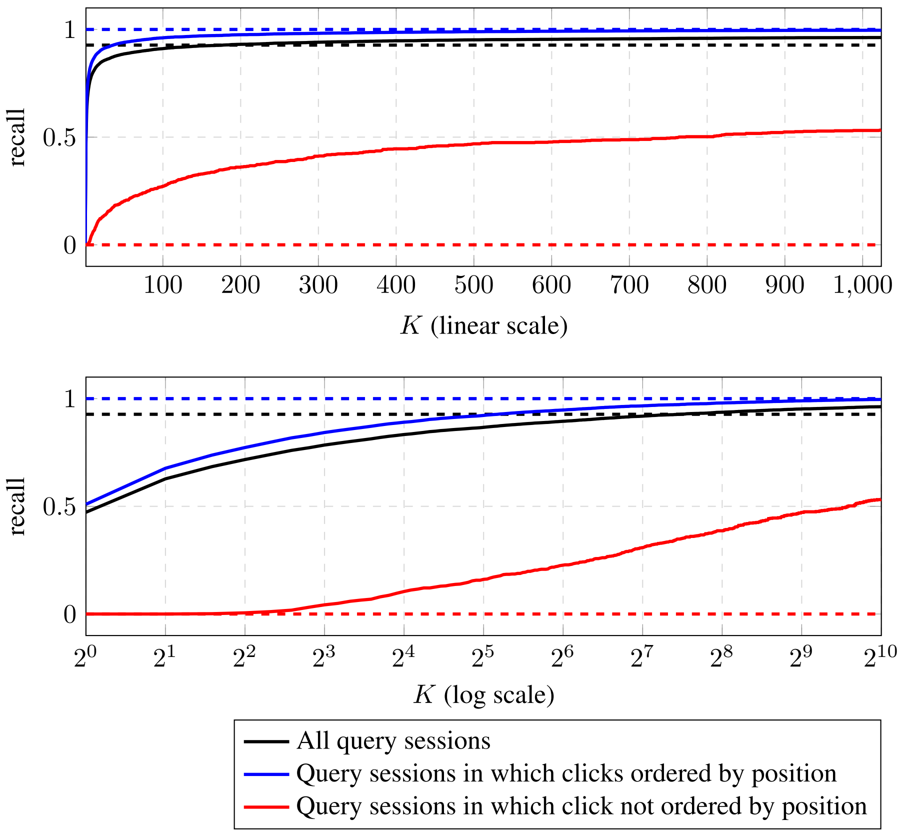


Figure 5.3: Recall at different values of K for (i) all query sessions (black, solid) (ii) query sessions in which clicks are ordered by position (blue, solid), and (iii) query sessions in which clicks are not ordered by position (red, solid). Dashed lines show percentages of query sessions in the corresponding groups, in which clicks on results happen in the order these results are presented on a SERP.

5.5.3 Experiment 1 (c)

Figure 5.4 shows recall at different values of K for (i) all query sessions and (ii) query sessions with L clicks. Dashed lines show percentages of query sessions in which clicks on results happen in the order these results are presented on a SERP.

For click sequences of length $L = 0$ and $L = 1$, recall of CSM approaches 1 (already for small values of K). Recall of CSM for sequences of length $L = 2, 3, 4$ at $K = 1024$ is higher than the percentages of query sessions in which click sequences of length L are ordered by position. For $L = 5$ and $K = 1024$, recall of CSM approaches the percentage of query sessions in which click sequences are of length 5 and are ordered by position. For sequences of length ≥ 2, recall of CSM first increases logarithmically with K (for $K \geq K_0(L)$), and then might increase both faster and

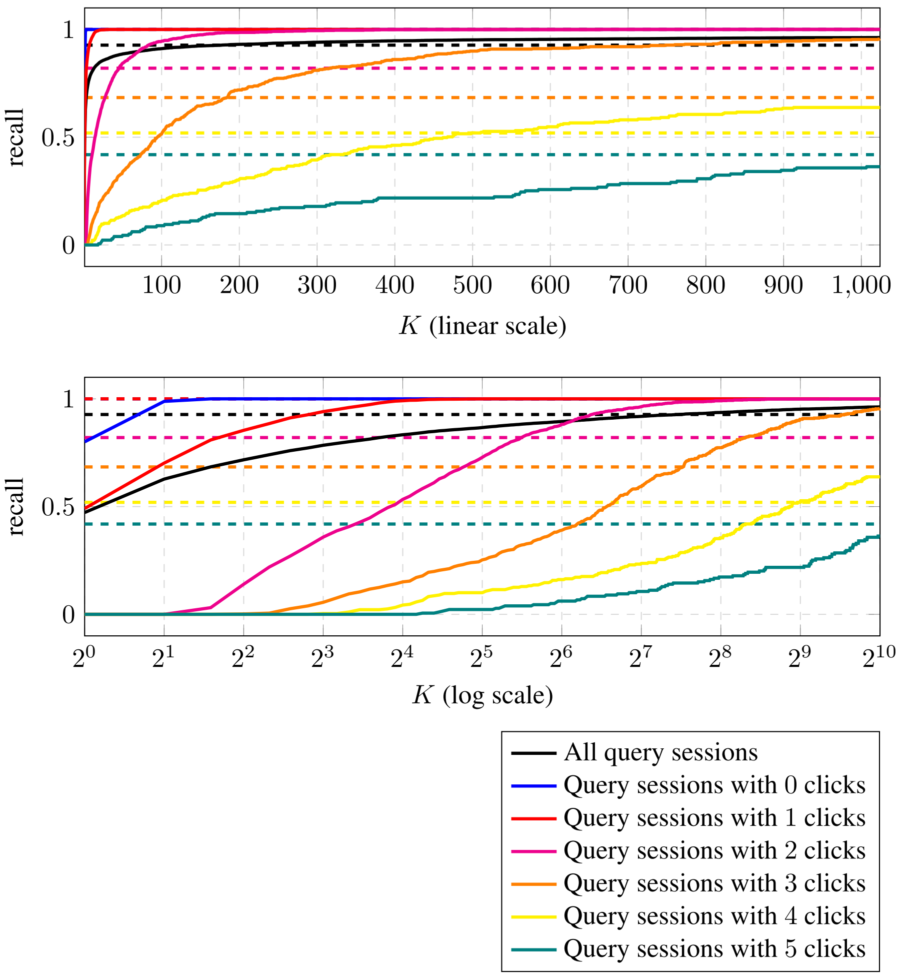


Figure 5.4: Recall at different values of K for (i) all query sessions and (ii) query sessions with L clicks. Dashed lines show percentages of query sessions in the corresponding groups, in which clicks on results happen in the order these results are presented on a SERP.

slower then logarithmically with K depending on L and the range of K.

We can see that the longer a click sequence is, the more difficult it is to predict such a sequence. This is intuitive and can be explained similarly to §5.5.2. First, we have more training data for shorter click sequences (see Table 5.1). Second, the number of possible click sequences of length L increases exponentially with L, making the prediction task more difficult.

Answering RQ 4.1 (c), we conclude that recall of CSM is very high in query sessions with a small number of clicks, and lower in query sessions with a larger number of clicks.

5.5.4 Experiment 1 (d)

Figure 5.5 plots, for different values of K, the total probability of K most probable click sequences predicted by CSM averaged over query sessions in the test set. We write $\sum_{i=1}^{K} P_{\text{CSM}}(s_i)$ to denote this probability.

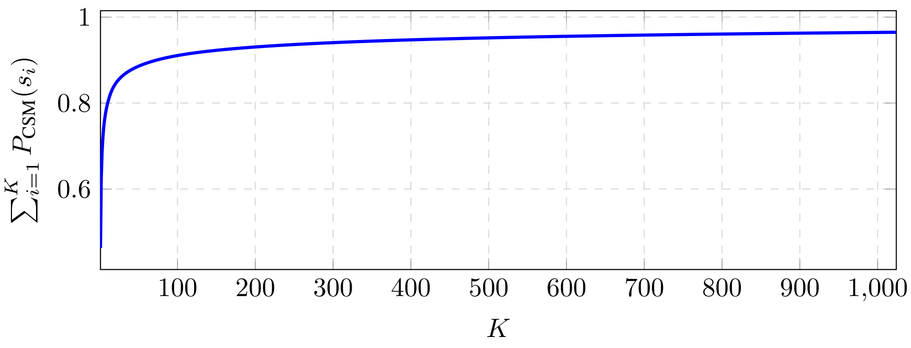


Figure 5.5: Total probability of the K most probable sequences predicted by CSM for different values of K.

We find that the total probability of K most probable click sequences grows fast with K, starting from 46.29% at $K = 1$ and 71.69% at $K = 4$, and achieving 91.81% at $K = 128$ and 96.47% at $K = 1024$. Thus, even for modest values of K, the total probability of the K most probable click sequences is not significantly less than 1.

Answering RQ 4.1 (d), we conclude that the K most probable click sequences according to CSM provide a good means to reason about the whole probability distribution over click sequences predicted by CSM.

5.5.5 Experiment 2

The results on Task 1 (§5.2.2) are given in Tables 5.2 and 5.3. Table 5.2 shows the perplexity of CSM upon observing a sequence of $\leq L$ clicks. Table 5.3 shows the AUC of CSM on the same task. Recall that the naive baseline predicts that a user will make $\leq L$ clicks with a constant probability optimized on the training set.

Table 5.2: Perplexity of observing a sequence of $\leq L$ clicks. Lower values correspond to better prediction performance.

	$L = 0$	$L \leq 1$	$L \leq 2$	$L \leq 3$	$L \leq 4$	$L \leq 5$
Baseline	1.8512	1.7169	1.4450	1.2779	1.1784	1.1068
CSM	1.7155	1.6153	1.3852	1.2438	1.1602	1.1029

Both Tables 5.2 and 5.3 show that CSM predicts the number of clicks better than the baseline and its performance increases with L (lower perplexity and higher AUC).

Table 5.3: AUC for the task of predicting whether a user will click on $\leq L$ results.

	$L = 0$	$L \leq 1$	$L \leq 2$	$L \leq 3$	$L \leq 4$	$L \leq 5$
Baseline	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
CSM	0.7362	0.7278	0.7353	0.7535	0.7566	0.7795

The latter result is intuitive as more sequences have $\leq L$ clicks for larger L and, thus, the prediction task becomes easier as L grows.

Answering RQ 4.2, we conclude that CSM provides a good means to predict the number of clicked results.

5.5.6 Experiment 3

The results on Task 2 (§5.2.2) are given in Table 5.4. The table shows the perplexity and AUC of CSM when predicting a sequence of clicks ordered by position.

Table 5.4: Performance for the task of predicting whether a user will click on results in the order these results are presented on a SERP. Lower values of perplexity and larger values of AUC correspond to better prediction performance.

	Perplexity	AUC
Baseline	1.2984	0.5000
CSM	1.2788	0.6826

Table 5.4 shows that CSM outperforms the baseline in terms of both perplexity and AUC. Thus, answering RQ 4.3, we conclude that CSM provides a good means to predict whether a user will interact with results in the order these results are presented on a SERP.

5.5.7 Experiment 4

The results on Task 3 (§5.2.2), i.e., the click prediction task, are given in Table 5.5.

Table 5.5 shows that CSM outperforms DBN, DCM, CCM and UBM by a large margin and matches the performance of NCM proposed in Chapter 4. Answering RQ 4.4, we conclude that CSM provides a good means to predict clicks on search engine results, achieving the state-of-the-art performance.

5.6 Related Work

We describe two types of related work: user interactions in search and user modeling.

Table 5.5: Perplexity for the click prediction task. Lower values correspond to better prediction performance.

Click model	Perplexity
DBN	1.3510
DCM	1.3627
CCM	1.3692
UBM	1.3431
NCM	1.3318
CSM	1.3312

5.6.1 User interactions in search

Log data from interactive systems such as search engines is one of the most ubiquitous forms of data available, as it can be recorded at little cost [164]. These data have become a popular source for improving the performance of search engines. In particular, logged interactions have been successfully adopted to improve various aspects of search, including document ranking [1, 80, 126], query auto-completion [79, 101] and query suggestion [25, 166], to improve recommender systems [124], optimizing presentations [161], and evaluation [67].

In the context of web search, many types of implicit information interaction behavior have been studied over the years. Early work, e.g., by Craswell et al. [36], focuses on single clicks and, in particular, on the *first* click. And assumes that a user abandons examination of web results upon the first click. Guo et al. [60] expand on this by studying sessions with *multiple* clicks, looking not just at the first click but also at follow-up clicks, the last click and dependencies between clicks, reflecting more complex information behavior.

There is a very broad spectrum of research that studies and tries to interpret information interaction behavior that involves multiple clicks, either by also taking additional signals into consideration or by zooming in on specific aspects of sequences of clicks. Examples of the former include work by Huurnink et al. [76] who examine click signals, download behavior, and purchase signals in vertical search and find high degrees of correlation between the three. Time, such as dwell time or time between user actions such as clicks, has been found to be another important source of implicit signals [16]: times elapsed between user actions provide a means to measure user satisfaction at the result level [48, 89], session level [48, 61] and system level [29, 141]. And beyond that, on mobile or screen-less devices there is range of interaction signals that are different from signals familiar from desktop environment – due to the context of use and due to gesture- and voice-based control, such as swipes, touch and voice conversations – and that have not been studied extensively [92]. Our work differs from these publications as we remain focused on click signals only and especially on sequences click signals.

Relevant examples of the studies that zoom in on specific aspects of multiple click behavior include work on repeat behavior such as repeated examinations or clicks [124, 170], which can be interpreted as strong evidence of the value ascribed to the result being examined or clicked again. In a similar vein, Scaria et al. [140] consider back clicks and last clicks; in their view, back clicks suggest a lack of progress on the current navigational path and, depending on contextual factors, last clicks mark success or failure. Hassan et al. [63] and Odijk et al. [125] focus on aspects of click sequences, including the number of clicks, their dwell time, and features to capture whether the user was clicking on the same results or results from the same domain multiple times, indicative of difficulty locating a particular resource. Williams and Zitouni [165] examine whether sequences of user interactions over time can be used to differentiate between good and abandonment and train an LSTM to distinguish between the two types of behavior. Especially relevant is the work by Wang et al. [157], who consider non-sequential examination and click behavior, both through an eye-tracking study and a log-based study. They arrive at several behavioral principles, for instance (i) between adjacent clicks, users tend to examine search results in a single direction without changes, and the direction is usually consistent with that of clicks; and (ii) although the examination behavior between adjacent clicks can be regarded as locally unidirectional, users may skip a few results and examine a result at some distance from the current one following a certain direction.

5.6.2 Modeling user interactions

To understand, describe and predict various types of user interactions discussed above, a number of user interaction models have been proposed aimed at modeling clicks [34], mouse movements [39], dwell time [89, 106], etc.

So far, modeling user clicks in search has attracted the most attention [34]. Click models usually represent clicks as binary random variables and construct a PGM that describes the dependencies between clicks and other (usually hidden) random variables, such as attractiveness (i.e., whether a snippet is attractive to a user given a query) and examination (i.e., whether a snippet is examined by a user) [27, 36, 46, 59, 60]. The advantage of PGM-based click models is that they intuitively describe user click behavior and can predict future clicks based on past observations [34]. Some click models take into account the order in which a user interacts with the results in order to better model and predict clicks [110, 157, 159, 168, 171]. However, such models either do not aim at predicting click sequences [110, 157, 159, 168] or consider only very short sequences of clicks [171].

Recently, neural click models have been proposed [17, 180]. The advantage of these models is that they do not require manually constructed PGMs to describe and predict user clicks, but rely on raw click data to learn hidden click patterns. Neural click models have better click prediction accuracy, but suffer from uninterpretability of the learned neural model as opposed to easily interpretable PGM-based click models. Also, as before, neural click models cannot predict sequences of clicks.

In addition to clicks, mouse movements between search results and various search-related timings have been studied and modeled. The probability of hovering over one element of a SERP after hovering over another element is predicted using the Farley-Ring model in [39]. Dwell time is modeled though Weibull and gamma distributions in [89, 106]. More timings, such as time between clicks, time to first/last click, etc., are considered in [16], where, in addition to the above distribution-based models, a context bias is modeled using neural networks.

What our work adds on top of the work listed above is our focus on *sequences* of clicks, and in particular on describing a set of *probably correct* click sequences.

5.7 Conclusions and Future Work

We studied the problem of predicting sequences of user interactions and, in particular, sequences of clicks. Specifically, we answered the following research question:

RQ 4 What are the challenges in predicting sequences of clicks and how to solve them?

We formally defined the problem of click sequence prediction and introduced the notion of probably correct click sequences. Furthermore, we proposed CSM, a neural network based model for predicting a probability distribution over click sequences. We advocated for using the K most probable click sequences predicted by CSM as a set probably correct click sequences. And we suggested to use these K click sequences to reason about the properties of the probability distribution over click sequences, such as the expected number of clicks and the expected order of clicks.

We evaluated the quality of CSM on a publicly available dataset. First, we showed that even for modest thresholds the K (larger than the threshold) most probable click sequences predicted by the CSM constitute a substantial part of the total probability mass assigned by the CSM to all possible click sequences, and thus can be regarded as probably correct click sequences predicted by CSM. We proposed to judge the success of a click sequence model $\mathcal{M}$ by the fact that the observed click sequence occurs in the list of the K most probable click sequences predicted by $\mathcal{M}$. We measured performance of CSM using recall@K, a metric that is also used to evaluate the performance of approximate nearest neighbor search methods [78, 120]. Our results showed that recall@K grows fast with K, starting from 47.24% at $K = 1$ and reaching 96.26% at $K = 1024$. We also found that recall@K increases slower with K in query sessions with larger number of clicks and in query sessions where users click on higher ranked results after clicking on a lower ranked result.

We also evaluated CSM on three prediction tasks: (i) predicting the number of clicks, (ii) predicting non-consecutive click sequences, and (iii) predicting clicks. The first two tasks were proposed in our work for the first time and the last one is a standard task used to evaluate click models. We found that CSM shows reasonable performance on the first two tasks, outperforming naive baselines that predict (i) that a user will click on $\leq L$ results, and (ii) that a user will click on the results in a non-consecutive order with a constant probability optimized on the training set. Finally, we observed that CSM reaches state-of-the-art performance on the task of predicting clicks, outperforming PGM-based click models DBN [27], DCM [60], CCM [60] and UBM [46] by a large margin, and matching the results of NCM proposed in Chapter 4, which is also implemented as a neural network.

In contrast to previous studies, which focus on modeling and predicting separate interaction events (e.g., a click on a result or mouse movement between two results) and properties of these separate events (e.g., time between clicks), our work focuses on understanding, modeling and predicting sequences of these events.

As to future work, we see two main directions: (i) to consider other representations of the user's query and the results returned by a search engine, and (ii) to extend CSM to non-linear SERP layouts. The user's query can be represented by its text, and the results by their content (title, snippet and main content). We believe that using content-based representations will allow us to learn more interesting dependencies between the results, and improve the performance for rare queries. The encoder proposed in §5.3.1 makes use of the fact that search results are presented as a list. Recommender systems present their results using non-linear layouts. Generalizing the encoder will make CSM suitable for applications outside of web search.

6 A Context-aware Time Model

In web search, information about times between user actions has been shown to be a good indicator of users' satisfaction with the search results. Existing work uses the mean values of the observed times, or fits probability distributions to the observed times. This implies a *context-independence* assumption that the time elapsed between a pair of user actions does not depend on the context, in which the first action takes place. We test this assumption using logs of a commercial web search engine and discover that it does not always hold. For between 37% to 80% of query-result pairs, depending on the number of observations, the distributions of click dwell times have statistically significant differences in query sessions for which a given result (i) is the first item to be clicked and (ii) is not the first. In this chapter, we answer the following research question asked in §1.1:

RQ 5 How to correctly interpret times between user actions observed in different contexts?

To account for the *context bias* effect mentioned above, we propose a *context-aware time model* (CATM). The CATM allows us (i) to predict times between user actions in contexts in which these actions were not observed, and (ii) to compute context-independent estimates of the times by predicting them in predefined contexts. Our experimental results show that the CATM provides a better means than existing methods to predict and interpret times between user actions.

6.1 Introduction

Search engine logs provide a rich source of information about user browsing behavior in web search. Recently, many models have been proposed to explain and predict user clicks on search engine results [34]. While click events are the most widely logged form of user interaction, there are also other behavioral signals that need to be understood, interpreted and modeled, and that can be used for ranking or prediction purposes.

We focus on behavioral signals based on times between user actions. The ability to accurately predict (i) click dwell time (i.e., time spent by a user on the landing page of a search result), and (ii) times from submission of a query to the first/last click on

This chapter is based on Borisov, Markov, de Rijke, and Serdyukov [16].

the results and to the next query submission (if none of the results will be clicked) allows us to optimize search engines for constructing result pages and suggesting query reformulations that minimize time it takes users to satisfy their information needs.

Existing work shows that times elapsed between user actions provide a means to measure user satisfaction at the result level [48, 89], session level [48, 61] and system level [29, 141]. To interpret times elapsed between user actions, existing work uses mean values of the observed times [1–3, 26, 29, 61–63, 107, 108, 125, 141], or fits probability distributions to the observed times [89, 106]. This implies a *context-independence assumption* that the time elapsed between a pair of user actions does not depend on the context in which the first action takes place.

We test this assumption by comparing the distributions of times associated with the same user action (i.e., submitting a given query or clicking on a given result presented for a given query), but preceded by different sequences of user interactions with the search engine. We find statistically significant differences between these distributions, which we explain by a *context bias effect*, not previously reported in the literature.

To account for the context bias effect, we propose a *context-aware time model* (CATM) that allows us to predict probability distributions of times between two user actions in a given context (which we represent by a sequence of previous user interactions with the search engine). The CATM can be used (i) to predict times between user actions in contexts, in which these user actions were not observed, and (ii) to compute context-independent estimates of the times by predicting them in predefined contexts. The *context-dependent* predictions can be used for personalized ranking and personalized query suggestion. The *context-independent* predictions can be used for predicting result relevance and in other tasks that use historical times between user actions.

We evaluate the CATM on four temporal prediction tasks (i.e., tasks in which we predict the time elapsed between two user actions), and find that the CATM outperforms the standard time modeling approach that fits probability distributions to the times observed for the first user action. We test the CATM's context-independent predictions of times between consecutive clicks on a ranking task (i.e., to rank a set of results by their relevance to a query), and find that the produced rankings result in better performance in terms of relevance than the rankings produced by the standard time modeling approaches.

In summary, we make the following contributions in this chapter.

C1 We introduce the notion of context bias in times elapsed between user actions, which has not previously been reported in the literature, and use logs of a commercial search engine to provide empirical evidence for it.

C2 We propose a context-aware time model. Through the use of contextual information (and, in particular, previous user interactions with a search engine), it provides a better means for predicting and interpreting times between user actions than existing time modeling techniques.

The rest of the chapter is organized as follows. In §6.2 we specify the considered temporal prediction tasks. In §6.3 we propose the context-aware time model. In §6.4 we detail our experimental setup and in §6.5 we present the outcomes of our experiments. We discuss related work in §6.6 and conclude in §6.7.

6.2 Temporal Prediction Tasks

We describe and motivate temporal prediction tasks that we consider in our work. In each task we aim to predict the time elapsed between a pair of user actions. The user actions can be of several types: submission of a query, click on a search result, click on an ad banner, scroll through search results, zoom in on a search result (in mobile search), etc. Here, we focus on user actions of the first two types: submission of a query (Q-action) and click on a search result (C-action), because these user actions occur in all search systems.

To define a temporal prediction task, we need to specify a set of action pairs between which we aim to predict times. When choosing these pairs, we follow two principles:

1. We select pairs of user actions for which the time elapsed between them is indicative of user satisfaction. The ability to predict times that help to anticipate user satisfaction with search results, allows us to construct result pages that maximize user satisfaction with the search.

2. We select pairs of user actions for which the times elapsed between them have a similar nature (i.e., similar human interpretations). This is important, because times that have different human interpretations are likely to be unequally useful in practical applications, and can also be used in different scenarios. Thus, we want to be able to measure the prediction performance of each group of times separately.

We define four temporal prediction tasks: *time-to-first-click*, *time-between-clicks*, *time-to-last-click*, and *time-from-abandoned-query*. The pairs of user actions used in each task are shown in Figure 6.1. Below, we describe them in more detail and provide

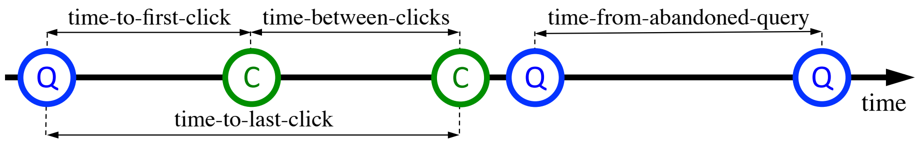


Figure 6.1: Times between pairs of user actions used in the considered temporal prediction tasks.

motivations for our focus on these specific tasks.

Time-to-first-click is the time between a submission of a query and the first click on search engine results (undefined if there were no clicks on search results). Existing work shows that this time reflects the quality of result presentation: the less time it takes a user to find an attractive result, the better in terms of user experience the search engine result page (SERP) is [134]. The ability to predict time-to-first-click allows us to construct SERPs that are more intuitive to users, i.e., require less time/effort to find relevant results. Practically speaking, the predicted time can be used in any application that uses the average time-to-first-click observed in search engine logs as its more precise alternative. Existing work uses the average time-to-first-click observed for a query-result pair as a feature to predict search task difficulty [3], search goal success [61, 62], urgent information needs (e.g., how to stem a severe bleeding) [119]; and the average time-to-first-click observed for a user to cluster users based on their SERP examination strategies [24].

Time-between-clicks is the time between two consecutive clicks on search engine results (undefined for the last click on search results). We use this time as a proxy for *click dwell time* (i.e., the time spent by a user on the landing page of a search engine result), which has been shown to be a good indicator of click satisfaction [89]. The ability to predict click dwell time allows us to construct SERPs that yield more satisfied clicks, e.g., by using the predicted time-between-clicks as a feature for the search result ranker. Agichtein et al. [1, 2] use average click dwell times to improve result relevance. The Yahoo! Learning to Rank dataset [26] uses them as features for ranking. The predicted dwell times can also be utilized in most other applications that make use of the average click dwell times observed in search engine logs, which include prediction of click satisfaction [89], result usefulness [107], search task difficulty [3, 108], search goal success [61, 62], struggling vs. exploring behavior [63, 125].

Time-to-last-click is the time between a submission of a query and the last click on search engine results (undefined if there were no clicks on search results). Radlinski et al. [134] show that this time reflects the quality of search engine results: the less time it takes a user to find a relevant result, the better in terms of user experience the search engine results are. The ability to predict time-to-last-click allows us to suggest queries that require less time (effort) to satisfy user's information needs. The predicted times can be used as features for query suggestion and for other applications such as prediction of search task difficulty and search goal success (playing a similar role as the average time-to-first-click).

Time-from-abandoned-query is the time from a submission of an *abandoned query* (i.e., a query with no clicks) to the next query submission. Song et al. [147] show that this time reflects the quality of search engine results in query sessions with no clicks (thus, it is complementary to time-to-last-click, which is defined in query sessions with at least one click). A short time interval indicates user dissatisfaction (negative abandonment): the user could quickly see the problem with the presented results and it took them little time to reformulate the query. A large time interval indicates user satisfaction (positive abandonment): the user most likely could find the answer on the SERP, and it took them some time to come up with the next query. The ability to predict time-from-abandoned-query allows us to suggest queries with high chances of positive abandonment and to avoid suggesting queries that are likely to be quickly reformulated. Similar to time-to-last-click, the predicted time-from-abandoned-query can be used for query suggestion, prediction of task difficulty, search goal success, etc.

There are several other times-between-user-actions that have previously been considered in the literature, but that we do not seek to predict in this chapter for the reasons discussed below.

Time-between-last-click-and-next-query is the time between the last click on search engine results presented for one query and the submission of the next query. Some previous work uses both this time and the time-between-clicks as a proxy for click dwell time (called *server-side click dwell time*) [88, 89]. While both times approximate click dwell time, there are important differences between them: time-between-clicks is a sum of the actual click dwell time and the time to choose the next result to click (which is typically small and exhibits low variance), while time-between-last-click-and-next-query is a sum of the actual click dwell time and the time to formulate the next query (which is sometimes large, and exhibits high variance). Because of these differences in nature, we do not want to model time-between-clicks and time-between-last-click-and-next-query together (see principle 2). We do not create a temporal prediction task for time-between-last-click-and-next-query, as it is very similar to the time-between-clicks task, but the observed times are likely to be more noisy.

Time-to-first-satisfied-click is the time between a submission of a query and the first click classified as satisfied. This time can be seen as a generalization of time-to-first-click and time-to-last-click: if we classify all clicks as satisfied, time-to-first-satisfied-click reduces to time-to-first-click; if we classify only the last click as satisfied, time-to-first-satisfied-click reduces to time-to-last-click. Most work defines satisfied clicks as clicks with dwell times larger than a predefined threshold (e.g., 30 seconds) [48]. Kim et al. [89] propose a method to adjust this threshold individually for each search result. We do not create temporal prediction tasks for all possible definitions of satisfied clicks, because we believe that the temporal prediction tasks for time-to-first-click and time-to-last-click capture the most important phenomena.

6.3 Method

Now that we have specified the temporal prediction tasks, we describe our approach to modeling times between user actions. In §6.3.1 we formalize the problem and introduce the notation. In §6.3.2 we describe two frameworks for modeling times between user actions. In §6.3.3 we describe the context-aware time model (CATM).

6.3.1 Problem statement

Let the sequence $(a_1, t_1), \ldots, (a_n, t_n)$ be a search history, which consists of user actions $a_1, \ldots, a_n$ and their timestamps $t_1, \ldots, t_n$. We aim to design a time model for predicting times $\tau_{i \to j} = t_j - t_i$ elapsed from the action a_i to the action a_j. We consider a probabilistic formulation of this problem, where the time $\tau_{i \to j}$ is a random variable with a probability density function $f(x; \theta)$, where θ denotes a set of parameters of the probability distribution.

To describe a probabilistic time model we need to specify a probability density function $f(x; \theta)$ and an algorithm to compute its parameters θ. Common choices for $f(x; \theta)$ are the exponential, gamma and Weibull probability density functions [89, 106]; see Table 6.1 for their parameterizations. Below, we discuss the frameworks for computing the function parameters θ.

6.3.2 Time modeling frameworks

We describe two frameworks for modeling times between user actions. The first framework, called *basic time modeling framework*, makes the context-independence assumption that time between a pair of user actions depends solely on the first action

Table 6.1: Probability density functions and their parameters.

Distribution	PDF $f(x;\theta)$	Parameters θ
Exponential	ae^{-ax}	$a > 0$
Gamma	$\frac{x^{a-1}e^{-x/b}}{b^a\Gamma(a)}$	$a, b > 0$
Weibull	$\frac{a}{b}\left(\frac{x}{b}\right)^{a-1} e^{-(x/b)^a}$	$a, b > 0$

ID, i.e., for submitting a query (Q-action), a unique identifier of the query string, and for clicking on a search result (C-action), an identifier of the query-result pair. The second framework, called *context-aware time modeling framework*, does not make the context-independence assumption and assumes that time between a pair of user actions depends on the first action ID and on the context in which the first action takes place.

Basic time modeling framework

Time models operating within the basic time modeling framework make the context-independence assumption that time $\tau_{i\to j}$ depends solely on the first action ID. We formalize this as $\tau_{i\to j} \sim f\left(x;\theta = \Theta(a_i)\right)$, where Θ denotes a mapping from the space of actions $\mathcal{A}$ to the space of parameters of the probability density function $f(x;\theta)$.[1] We learn the mapping Θ by maximizing the likelihood of the observed times $\tau_{i\to j}$. The corresponding optimization problem can be formalized as follows:

$$\begin{aligned} \text{DATA} &= \{(a_i, \tau_i) \mid a_i \in \mathcal{A}, \tau_i \in [0;\infty)\}_{i=1}^{N} && (6.1)\\ \tau_i &\sim f(x;\theta = \Theta(a_i)) && (6.2)\\ \widehat{\Theta} &= \arg\max_{\Theta} \prod_{i=1}^{N} f(\tau_i;\theta = \Theta(a_i)). && (6.3) \end{aligned}$$

Here, we write N to denote the total number of observations, $\widehat{\Theta}$ to denote the solution of this optimization problem, and we also change τ indices to simplify the notation.

Without any constraints on the mapping $\widehat{\Theta}$, this optimization problem can be further decomposed into a set of per action ID *maximum likelihood estimation* (MLE) problems (we apply the logarithm function, which is monotonic and therefore does not change the result of the $\arg\max$ operator):

$$\widehat{\Theta}(a) = \arg\max_{\theta} \sum_{i=1}^{N} \mathbf{I}_a(a_i) \log f(\tau_i;\theta), \tag{6.4}$$

where $\mathbf{I}_a(x)$ denotes the indicator function, i.e., 1 if $x = a$ and 0 otherwise. For the exponential distribution, the MLE of its parameter (Table 6.1) is the inverse of the mean

[1] Where no confusion can arise, we write a_i to denote both the action performed by a user and the action ID.

of the observed times. For the gamma and Weibull distributions, there is no closed form solution for the MLE problem. But it is possible to express one of their parameters as a function of the other one and $\tau_1, \dots, \tau_N$, and to reduce the MLE problem to minimization of a scalar function.

Existing work on modeling click dwell times [89, 106] operates within the basic time modeling framework and follows the described approach to estimate their parameters.

Context-aware time modeling framework

We argue that considering only the ID of the first action is not enough to accurately model the time $\tau_{i\to j}$, and propose, in addition to using the ID of a_i, to use the context c_i in which the action a_i takes place. We formalize this as $\tau_i \sim f\left(x; \theta = \Theta(a_i, c_i)\right)$, where Θ denotes a mapping from the Cartesian product of the space of actions $\mathcal{A}$ and the space of contexts $\mathcal{C}$ to the parameters of the probability density function $f(x;\theta)$.

Similar to Eqs. 6.1, 6.2 and 6.3, the optimization problem in the context-aware time modeling framework can be formalized as follows:

$$\begin{aligned} \text{DATA} &= \{(a_i, c_i, \tau_i) \mid a_i \in \mathcal{A}, c_i \in \mathcal{C}, \tau_i \in [0;\infty)\}_{i=1}^{N} && (6.5)\\ \tau_i &\sim f(x; \theta = \Theta(a_i, c_i)) && (6.6)\\ \widehat{\Theta} &= \arg\max_{\Theta} \prod_{i=1}^{N} f(\tau_i; \theta = \Theta(a_i, c_i)). && (6.7) \end{aligned}$$

However, without any additional constraints on the mapping $\widehat{\Theta}$, the solution of this optimization problem will have very poor generalization due to the sparsity of action-context pairs (a_i, c_i). Let us illustrate this using a few simple definitions of the context c_i: (i) number of previously clicked results, (ii) positions of previously clicked results, (iii) positions of previously clicked results and times between previous clicks. For these definitions of the context c_i, we list in Table 6.2 the number of unique action-context pairs (a_i, c_i) for which we need to estimate parameters θ. We see that the number of $\widehat{\Theta}$

Table 6.2: Number of unique action-context pairs (a_i, c_i) for different definitions of the context c_i; N denotes the number of results on a SERP; T denotes the number of time intervals we consider.

Context c_i	# of unique (a_i, c_i)
None	$\mathcal{O}(\lvert\mathcal{A}\rvert)$
Number of previously clicked results	$\mathcal{O}(\lvert\mathcal{A}\rvert \times N)$
Positions of previously clicked results	$\mathcal{O}(\lvert\mathcal{A}\rvert \times N^N)$
Positions of previously clicked results and times between previous clicks	$\mathcal{O}(\lvert\mathcal{A}\rvert \times (N \times T)^N)$

parameters becomes too large to be able to estimate them using the amount of log data that can be generated by search engines.

The natural way to handle this problem is to constrain the family of mappings $\widehat{\Theta}$ in Eq. 6.7. In §6.3.3, we describe the *context-aware time model* that operates within the proposed context-aware time modeling framework and provides a way to constrain the family of mappings $\widehat{\Theta}$.

6.3.3 Context-aware time model

In §6.3.2 we introduced the context-aware time modeling framework that models times between two user actions using the ID of the first action and the context in which it takes place. To construct a model that operates within this framework, we need to specify (i) the probability density function $f(x;\theta)$, (ii) representation of the context c_i, and (iii) constraints on the mapping $\widehat{\Theta}$ (Eq. 6.7). We also need to describe the solution of the optimization problem (6.5)–(6.7) for the chosen constraints.

Probability density function $f(x;\theta)$

We follow previous work [89, 106] and use the exponential, gamma and Weibull probability density functions.

Context c_i

We represent the context c_i, in which the action a_i takes place, by a sequence of user interactions with a search engine that preceded the action a_i. We list the attributes that we use to describe the user interactions in Table 6.3.

Table 6.3: Attributes we use to describe user interactions with a search engine.

General	
Is Q-action	(0: no, 1: yes)
Is C-action	(0: no, 1: yes)
$\log(1 +$ observed time since previous action$)$	(0: undefined)
$\log(1 +$ average time since previous action$)$	(0: undefined)
Q-action	
Is new search session	(0: no, 1: yes)
Number of terms in issued query	(0: undefined)
BM25 (issued query, previous query)	(0: undefined)
BM25 (previous query, issued query)	(0: undefined)
C-action	
Is click on the 1st position	(0: no, 1: yes)
...	...
Is click on the 10th position	(0: no, 1: yes)

The first two attributes from the general section distinguish between Q- and C-actions. The third and fourth attributes inform us about the actual times observed between the previous actions and the average times between them. This information is useful to account for the user's reading speed, persistence, etc.

The first attribute from the Q-action section informs us about whether the query is the first in a search session or not. It is useful to distinguish between the cases (i) when a user has just started browsing, and (ii) when a user has been browsing for some time. In §6.5.1 we show that these cases may result in different probability distributions of times from submitting a query to the first click on the results, last click on the results and next query submission (if none of the results will be clicked). The second, third and fourth attributes from the Q-action section inform us about the query's similarity with the previous query.

The attributes from the C-action section inform us about the position of the clicked result. Existing work shows that the position of a result influences users' trust in the result's usefulness, which affects their decisions whether to click or not [34]. We allow for the possibility that the result's position might affect the time from clicking on the result to other actions.

Constraints on the mapping $\widehat{\Theta}(a_i, c_i)$

Without any constraints on the mapping $\widehat{\Theta}$, the solution of the optimization problem (6.5)–(6.7) will have very poor generalization due to a very large number of action-context pairs (see §6.3.2). To deal with this problem we impose additional constraints on the family of mappings $\widehat{\Theta}$. In particular, we limit the family of possible mappings $\widehat{\Theta}(a_i, c_i)$ to the ones that can be decomposed into the mappings $\widehat{\Theta}_A(a_i)$ and $\widehat{\Theta}_C(c_i)$ defined on the space of actions $\mathcal{A}$ and the space of contexts $\mathcal{C}$, respectively. Not only does this reduce the number of effective $\widehat{\Theta}$ parameters from $\mathcal{O}(|\mathcal{A}| \times |\mathcal{C}|)$ to $\mathcal{O}(|\mathcal{A}| + |\mathcal{C}|)$, but it also separates action-specific and context-specific parameters. The latter allows us (i) to estimate times between user actions in contexts in which these user action were not observed; and (ii) to estimate context-independent parameters of times between actions, which can be used for ranking and other prediction tasks that use historical times between user actions.

Now that we have described the constraints on the mapping $\widehat{\Theta}$ at the functional level, we need to explain the mapping $\widehat{\Theta}_A$, the mapping $\widehat{\Theta}_C$, and the decomposition of the mapping $\widehat{\Theta}$ into the mappings $\widehat{\Theta}_A$ and $\widehat{\Theta}_C$ in more detail.

Mapping $\widehat{\Theta}_A$. The mapping $\widehat{\Theta}_A$ is an equivalent of the mapping Θ in the basic time modeling framework. As in that setting, we treat it as a table of per action ID parameters, which we call *context-independent* parameters of the function $f(x; \theta)$. Thus, the number of parameters of the mapping $\widehat{\Theta}_A$ is $\mathcal{O}(|\mathcal{A}|)$.

Mapping $\widehat{\Theta}_C$. The mapping $\widehat{\Theta}_C$ describes how to adjust context-independent parameters computed by the mapping $\widehat{\Theta}_A$ to account for the context in which the first action takes place. We want this mapping to have $\mathcal{O}(|\mathcal{A}|)$ parameters so as not to increase the asymptotic number of parameters compared to time models that operate within the basic time modeling framework. Implementing the mapping $\widehat{\Theta}_C$ as a table of per context parameters will lead to a total of $\mathcal{O}(|\mathcal{C}|)$ parameters, which is likely to dominate $\mathcal{O}(|\mathcal{A}|)$. Thus, we decide to implement the mapping $\widehat{\Theta}_C$ using a machine learning algorithm. In particular, we implement it as a *recurrent neural network* (RNN) with *long short-term memory* (LSTM) cells [66]. We choose this architecture, because we represent the context c_i as a sequence of numerical attributes (see §6.3.3); and for tasks that involve processing sequential data, RNNs have demostrated strong performance on a wide range of tasks. See examples in language modeling [117], speech recognition [56] and machine translation [148]. By using an RNN we reduce the number of $\widehat{\Theta}$ parameters to $\mathcal{O}(N \times M)$, where N denotes the number of attributes we use to represent user interactions with the search engine (18 in our work) and M denotes the maximum number of units in the RNN layers (256 in our work). As $N \times M \ll |\mathcal{A}|$, we satisfy the $\mathcal{O}(|\mathcal{A}|)$ requirement.

Decomposition of $\widehat{\Theta}$ into $\widehat{\Theta}_A$ and $\widehat{\Theta}_C$. A decomposition of $\widehat{\Theta}$ into $\widehat{\Theta}_A$ and $\widehat{\Theta}_C$ describes how to compute the parameters θ of the probability density function $f(x;\theta)$ from the action-specific parameters $\widehat{\Theta}_A(a_i)$ and the context-specific parameters $\widehat{\Theta}_C(c_i)$. We want the action-specific parameters $\widehat{\Theta}_A(a_i)$ to be context-independent estimates of parameters of the probability density function $f(x;\theta)$ for the action ID a_i (see the discussion above), and the context-specific parameters $\widehat{\Theta}_C(c_i)$ to be coefficients that inform us about how to adjust $\widehat{\Theta}_A(a_i)$ to account for the context c_i. One natural way to achieve it is (i) to have two context-specific parameters α and β for each action-specific parameter θ_i, and (ii) to apply a linear transformation $g(\theta_i) = \alpha\theta_i + \beta$ to compute the context-dependent parameters of the probability density function. We formalize it as:

$$\widehat{\Theta}(a_i, c_i) = \widehat{\Theta}_A(a_i) \circ \widehat{\Theta}^0_C(c_i) + \widehat{\Theta}^1_C(c_i). \tag{6.8}$$

Here, $\widehat{\Theta}^0_C(c_i)$ and $\widehat{\Theta}^1_C(c_i)$ constitute the first and the last halves of the vector $\widehat{\Theta}_C(c_i)$, which contains two times more elements than $\widehat{\Theta}_A(a_i)$, and the symbol $\circ$ denotes the element-wise product.

Learning the mappings $\widehat{\Theta}_A(a_i)$ and $\widehat{\Theta}_C(c_i)$

Now, we describe an approximate solution of the optimization problem (6.5)–(6.7) with the constraints on the mapping $\widehat{\Theta}$ defined in §6.3.3. This problem does not have closed form solutions for the mappings $\widehat{\Theta}_A$ and $\widehat{\Theta}_C$. Therefore, we propose an iterative algorithm to compute them (see Algorithm 1). We use $\widehat{\Theta}(a_i, c_i) = \mathcal{F}(\widehat{\Theta}_A(a_i), \widehat{\Theta}_C(c_i))$ as a generalized version of Eq. 6.8.

The algorithm initializes the mapping $\widehat{\Theta}_A$ with the solution of the optimization problem (6.1)–(6.3) in the basic time modeling framework (line 1). Then it alternates between learning $\widehat{\Theta}_C$ while fixing $\widehat{\Theta}_A$ (line 3) and learning $\widehat{\Theta}_A$ while fixing $\widehat{\Theta}_C$ (line 4) until convergence. In our experiments the process converges after 5–10 iterations (i.e., passes through the loop). It is easy to show that the likelihood does not decrease between the iterations.

To find the best $\widehat{\Theta}_C$ while fixing $\widehat{\Theta}_A$ (Algorithm 1, line 3), we use the *stochastic gradient descent* (SGD) algorithm with minibatches, because this is the most widely used algorithm for training neural networks [8]. The learning rates for each parameter are adjusted according to the ADADELTA algorithm [177] (we use the default values of $\epsilon = 10^{-6}$ and $\rho = 0.95$). We also use the gradient clipping technique [130] to alleviate the exploding gradient problem [9] (we set the value of the threshold to 1).

To find the best $\widehat{\Theta}_A$ while fixing $\widehat{\Theta}_C$ (Algorithm 1, line 4), we decompose the

Algorithm 1 Learn $\widehat{\Theta}_A(\cdot), \widehat{\Theta}_C(\cdot)$

```
1: $\widehat{\Theta}_A \leftarrow \arg\max_{\Theta} \prod_{i=1}^{N} f(\tau_i; \theta = \Theta(a_i))$
2: while not $\widehat{\Theta}_A, \widehat{\Theta}_C$ converged do
3:     $\widehat{\Theta}_C \leftarrow \arg\max_{\Theta_C} \prod_{i=1}^{N} f(\tau_i; \theta = \mathcal{F}(\widehat{\Theta}_A(a_i), \Theta_C(c_i))$
4:     $\widehat{\Theta}_A \leftarrow \arg\max_{\Theta_A} \prod_{i=1}^{N} f(\tau_i; \theta = \mathcal{F}(\Theta_A(a_i), \widehat{\Theta}_C(c_i))$
5: end while
6: return $\widehat{\Theta}_A, \widehat{\Theta}_C$
```

optimization problem into a set of per action ID MLE problems (similar to Eq. 6.4):

$$\widehat{\Theta}_A(a) = \arg\max_{\theta} \sum_{i=1}^{N} \mathbf{I}_a(a_i) \log f(\tau_i; \theta = \mathcal{F}(\theta, \widehat{\Theta}_C(c_i)).$$

We solve these MLE problems using the *limited memory BFGS with bound constraints (L-BFGS-B)* algorithm [181]. We use bound constraints to ensure that the distribution parameters θ take admissible values (e.g., the values of the parameters of the exponential, gamma and Weibull distributions need to be positive).

To summarize, we predict a probability distribution over the time elapsed between a pair of user actions. Unlike existing methods, which rely solely on the first action ID, CATM also considers the context in which the first action takes place. CATM represents this context as a sequence of user interactions with a search engine that precede the first action, and employs a recurrent neural network that learns how to adjust the first action ID parameters using this context representation. CATM can be used (i) to predict the time elapsed between a pair of user actions in a context, in which these actions have not been previously observed, and (ii) to obtain a context-independent estimate of the time between two user actions by predicting it in a predefined context.

6.4 Experimental Setup

In this section we describe our experimental setup. We start with the research questions that we seek to answer (§6.4.1). Then we describe the datasets that we use (§6.4.2) and the evaluation methodology that we follow (§6.4.3). Finally, we describe the experiments that we conduct to answer our research questions (§6.4.4).

6.4.1 Research questions

We split RQ 5 into three subquestions:

RQ 5.1 Can we observe the context bias effect? More precisely, can we observe a difference in time probability distributions for different contexts in which the first action takes place?

RQ 5.2 Do the context-aware time models, which besides the first action ID make use of its context, provide a better means to explain times between user actions than the basic time models, which make the context-independence assumption and rely solely on the first action ID?

RQ 5.3 Do the context-independent predictions of the CATMs, trained on the time-between-clicks task, provide a better means to rank search results than existing methods based on the observed times between clicks?

6.4.2 Dataset

To construct datasets for the temporal prediction tasks described in §6.2, we collected three months of log data from a commercial web search engine. We use the first two months of the log data to train time models and the last month to evaluate their prediction performance. In each dataset, we filter out times associated with actions IDs that occur less than 25 times in the training set. For time-to-first-click and time-from-abandoned-query, we also filter out times larger than 1 minute; for time-to-last-click and time-between-clicks, we filter out times larger than 5 minutes (this complies with actual times reported in [134]). The number of observations in the resulting datasets are shown in Table 6.4.

Table 6.4: Number of observations in the datasets for each temporal prediction task.

Time between actions	Number of observations
Time-to-first-click	30,747,733
Time-between-clicks	6,317,834
Time-to-last-click	30,446,973
Time-from-abandoned-query	11,523,351

To compare the performance of ranking models based on time-between-clicks, we also collected relevance labels for 50,137 query-result pairs that occur in our time-between-clicks dataset. The relevance labels were assigned by trained judges on a scale from 0 to 4, with 0 = bad, 1 = fair, 2 = good, 3 = excellent, 4 = perfect.

6.4.3 Evaluation methodology

We evaluate the performance of time models using the *log-likelihood* and the *root mean squared error* (RMSE) metrics. The log-likelihood metric shows how well a time model explains the observed times between user actions. We report the logarithm of the likelihood function, averaged over all observations in the test set. Larger values of the metric indicate better performance. The RMSE metric shows the average difference between the expected values of the time probability distributions predicted by a model and the observed times-between-actions. Lower values of the metric indicate better performance. We also evaluate the time models for predicting time-between-clicks on a ranking task (i.e., to rank a set of results by their relevance to a query). We use the NDCG metric [77], and report its scores at truncation levels 1, 3, 5 and 10. Larger values of the metric indicate better performance.

We perform significance testing in terms of all metrics using a paired t-test; the differences are considered statistically significant for p-values lower than 0.05.

6.4.4 Experiments

Experiment 1. To answer RQ 5.1, for each action ID we split the observed times in two *context groups*, which correspond to different sets of previous user interactions, and run the two-sample two-sided Kolmogorov-Smirnov (KS) test [35] to determine whether the observed times were drawn from the same distribution. The null hypothesis states that the observed times were drawn from the same distribution, which means that there is no context bias effect. Rejecting it, at least for some action IDs, will prove the existence of the context bias effect for these action IDs. Not being able to reject it might happen for several reasons. First, the context bias effect may not appear for certain types of queries (e.g., navigational queries) and results (e.g., irrelevant results). Second, we may not have enough data to reject the null hypothesis. Third, we may not have chosen the best context groups.

When choosing a rule to split times-between-actions in two context groups based on the context, we give preference to easily interpretable and easily reproducible ones. For the time-to-first-click, time-to-last-click and time-from-abandoned-query, we split the observed times based on whether the query is the first in the search task or not. We say that a query is the first in the search task if it does not share terms with the previous query in the search session, or if it is the first query in the search session. For the time-between-clicks, we split the observed times based on whether the clicked result is the first item to be clicked on SERP or not.

We use the KS test, because the more popular t-test is not applicable in our setting: it requires the tested samples to be drawn from a normal distribution, which is not the case for time observations. An alternative to the KS test could be the Mann-Whitney U test [35], but following previous work [106] we use the KS test.

Without a sufficient number of observations in both context groups, the KS test would be unable to detect a difference between samples even if one exists. For this reason, we apply the KS test only to actions IDs, for which there are, at least, N observations in both context groups. Using a large value of N improves test sensitivity (increases the number of action IDs for which the null hypothesis can be rejected), but reduces the number of action IDs that we use in our experiment. Thus, we run our experiment for different values of N. In particular, we use N in the range $[25, 200]$. Table 6.5 shows the number of action IDs with, at least, N observations in both context groups for $N = \{25, 50, 100, 200\}$.

Experiment 2. To answer RQ 5.2, we compare the performance in terms of log-likelihood and RMSE of the context-aware time models against the basic time models on the four temporal prediction tasks described in §6.2.

Table 6.5: Number of action IDs with at least $N = \{25, 50, 100, 200\}$ observations in each context group for different times-between-actions.

	N			
Time between actions	25	50	100	200
Time-to-first-click	60,284	24,368	7894	1545
Time-between-clicks	18,954	8335	3289	1288
Time-to-last-click	59,367	23,956	7756	1500
Time-from-abandoned-query	22,190	9180	2980	569

Experiment 3. To answer RQ 5.3, we compare the performance in terms of NDCG of rankings models based on (i) the average values of the observed times between clicks, (ii) the expected values of the probability distributions fitted to the observed times between clicks, and (iii) the expected values of the context-independent probability distributions predicted by the CATMs trained on the time-between-clicks task. To predict the context-independent probability distributions we use the following fixed context: (i) the query is the first in the search session; (ii) the result is presented on the top position and is the first item to be clicked; (iii) the time elapsed between the query submission and the click on the result is 4 seconds (the median of the times-to-first-click observed in our dataset).

6.5 Results

In this section we present the results of the experiments described in §6.4.4 and provide answers to the research questions stated in §6.4.1.

6.5.1 Experiment 1

The results of Experiment 1 are given in Figure 6.2. The figure shows the percentage of action IDs in each temporal prediction task, for which the Kolmogorov-Smirnov test rejected the null hypothesis ($p < 0.05$) in favor of the alternative hypothesis, which states that the times-between-actions in the chosen context groups were drawn from different probability distributions. Equivalently, the alternative hypothesis states that the context bias effect exists.

RQ 5.1. Figure 6.2 shows that for each time-between-actions there is more than 5% action IDs, for which the null hypothesis is rejected. This proves the existence of the context bias effect. Indeed, if times-between-actions did not depend on previous user interactions (null hypothesis was true), the number of action IDs for which the null hypothesis would be rejected had to be 5% (significance level) of the total number of the tested action IDs.

The percentage of action IDs for which the context bias effect is detected, increases with the minimum number of observations in each context group (Figure 6.2). This suggests that the actual number of action IDs for which the context bias effect appears,

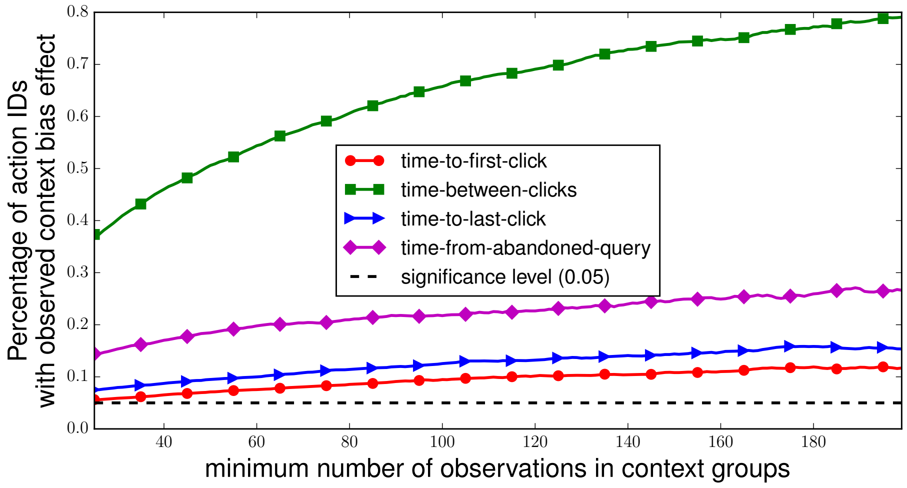


Figure 6.2: Percentage of actions IDs with observed context bias effect vs. minimum number of observations in context groups.

is even larger than the number of action IDs for which we manage to detect the effect. And with more interaction data, it should be possible to detect the effect in a larger number of times-between-actions.

From the above results we conclude that there is a tangible context bias effect, which results in statistically significant differences in time-between-actions probability distributions for different contexts (in our case, for different sets of previous user interactions).

6.5.2 Experiment 2

The results of Experiment 2 are given in Table 6.6. The table shows the performance of the basic time models and the context-aware time models in terms of log-likelihood and RMSE on four temporal prediction tasks: time-to-first-click, time-between-clicks, time-to-last-click and time-from-abandoned-query (§6.2).

RQ 5.2. Table 6.6 shows that the context-aware time models outperform the basic time models in terms of both the log-likelihood and RMSE metrics on all temporal prediction tasks. The improvements for each task and each distribution are statistically significant with $p < 0.001$. The improvements in terms of log-likelihood are comparable with the differences between the basic time models using different probability density functions, and in most cases exceed them. The improvements in terms of RMSE vary, depending on the task, between 0.98–5.32%.

To further understand the performance difference between the basic time models and the context-aware time models, we plot their performance vs. the actual times-between-actions observed in the datasets. Figures 6.3 and 6.4 show the performance on the time-between-clicks task. We start with Figure 6.3, which shows the performance in terms of the log-likelihood metric. Here, similar to the differences between the

Table 6.6: Performance of the basic time models and the context-aware time models on four temporal prediction tasks. Larger values of the average log-likelihood metric and lower values of the root mean squared error (RMSE) metric indicate better performance. Improvements of the context-aware time models over the basic time models using the same probability density functions (exponential, gamma, Weibull) are statistically significant ($p < 0.001$) in terms of both metrics.

Time model	Distribution	Log-likelihood	RMSE
	Time-to-first-click		
Basic	exponential	-2.9050	7.52
	gamma	-2.8399	7.51
	Weibull	-2.8970	7.52
Context-aware	exponential	-2.8715	7.29
	gamma	-2.7636	7.25
	Weibull	-2.8449	7.27
	Time-between-clicks		
Basic	exponential	-4.9219	60.73
	gamma	-4.9105	60.76
	Weibull	-4.9077	60.76
Context-aware	exponential	-4.8787	58.93
	gamma	-4.8556	58.98
	Weibull	-4.8504	58.94
	Time-to-last-click		
Basic	exponential	-3.8360	40.85
	gamma	-3.7849	40.85
	Weibull	-3.7386	40.85
Context-aware	exponential	-3.7924	40.45
	gamma	-3.7456	40.45
	Weibull	-3.7073	40.51
	Time-from-abandoned-query		
Basic	exponential	-3.5862	12.41
	gamma	-3.5422	12.41
	Weibull	-3.5560	12.41
Context-aware	exponential	-3.5210	11.75
	gamma	-3.4235	11.76
	Weibull	-3.4554	11.77

basic time models using different probability density functions, we observe major improvements of the context-aware time models over the basic time models for short times. In Figure 6.4, which shows the performance in terms of the RMSE metric, we

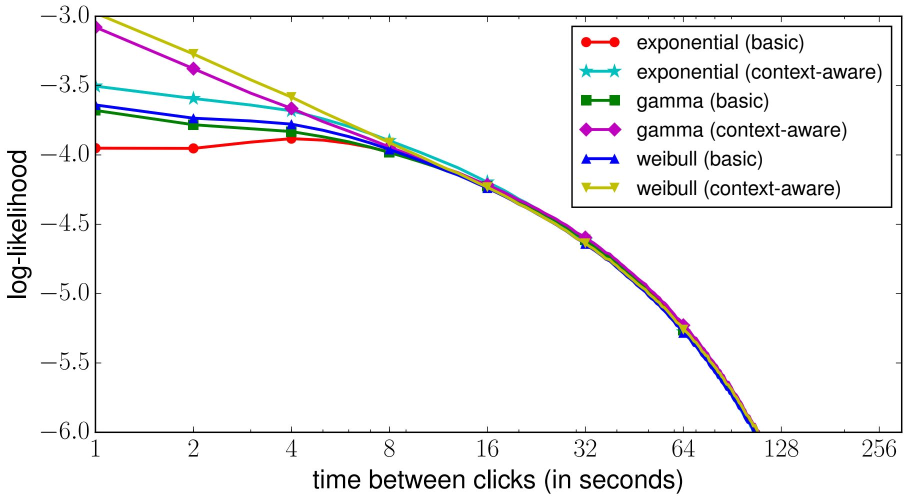


Figure 6.3: Time model performance in terms of the log-likelihood metric on the time-between-clicks task vs. actual times observed in the dataset.

also observe major improvements for short times.

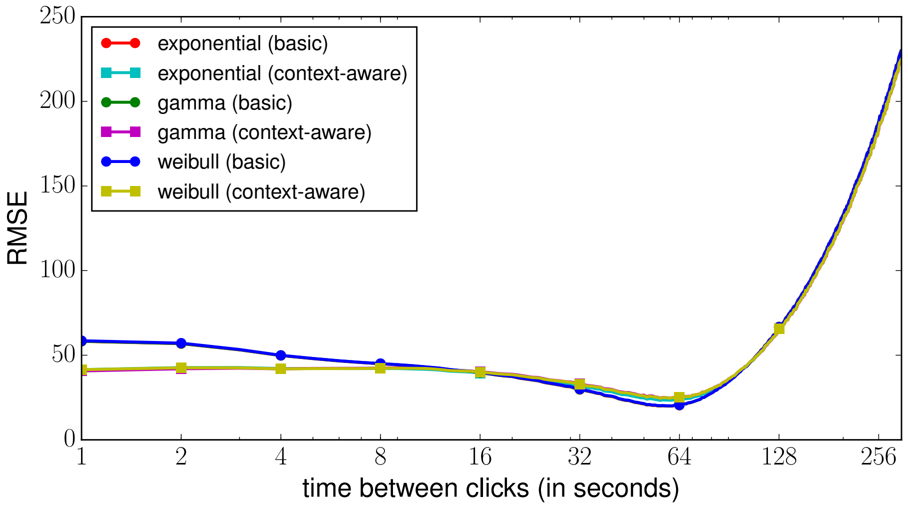


Figure 6.4: Time model performance in terms of the RMSE metric on the time-between-clicks task vs. actual times observed in the dataset.

A reader might notice a drop in performance of the context-aware time models compared to the basic time models between 17 and 99 seconds in Figure 6.4. This is better seen in Figure 6.5, which plots the differences between the context-aware time models and the basic time models shown in Figure 6.4. Here, we see that the context-

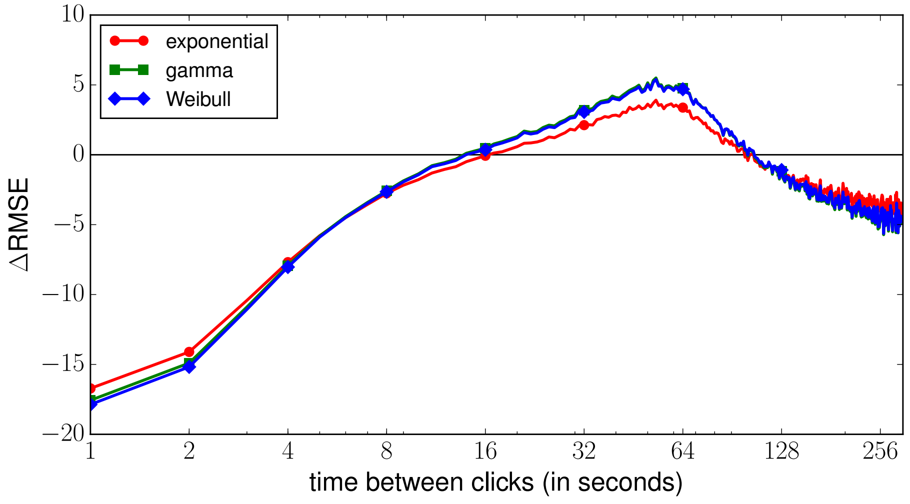


Figure 6.5: Performance difference in terms of the RMSE metric between the context-aware time models and the basic time models on the time-between-clicks task vs. actual times observed in the dataset.

aware time models perform better for times shorter than 17 seconds and longer than 99 seconds, and perform worse for times in the range of 17–99 seconds. This can be explained as follows. The average time-between-clicks in our dataset is 53.44 seconds. A naive time model that always predicts time-between-clicks to be 53.44 seconds, would have very low RMSE around 53.44 seconds and high RMSE for shorter and longer times (the overall RMSE performance of this naive approach would be low). The basic time models predict time-between-clicks across the whole time range including short and long times, and have better overall RMSE performance. However, this is achieved at the cost of having higher RMSE around the average time-between-clicks (i.e., 53.44 seconds) compared to that of the naive time model. The proposed context-aware time models, on average, predict time-between-clicks better than the basic time models (and especially so for shorter and longer times, see Figure 6.5). This is again achieved at the cost of having higher RMSE around the average time-between-clicks (i.e., 53.44 seconds). In fact, shorter times-between-clicks usually correspond to dissatisfied clicks, and longer times-between-clicks correspond to satisfied clicks [48]. Thus, in order to improve user satisfaction with the search, it is more important to predict short and long times-between-clicks rather than to distingush between times close to the average time-between-clicks.

From the above results we conclude that the context-aware time models, which besides the first action ID make use of its context, provide a better means to explain times-between-actions that the basic time models, which rely solely on the first action ID.

6.5.3 Experiment 3

Table 6.7 shows the outcomes of Experiment 3, a comparison of the performance of ranking models based on times-between-clicks.

Table 6.7: Performance of ranking models based on the time between clicks. Larger values of the NDCG metric correspond to better rankings. The improvements of the context-aware time models over the existing methods are statistically significant ($p < 0.05$).

		NDCG			
Time model	Distribution	@1	@3	@5	@10
Average	n/a	0.651	0.693	0.728	0.812
Basic	exponential	0.651	0.693	0.728	0.812
	gamma	0.646	0.693	0.728	0.812
	Weibull	0.656	0.699	0.730	0.811
Context-aware	exponential	0.668	0.710	0.743	0.820
	gamma	0.675	0.715	0.748	0.822
	Weibull	0.671	0.709	0.745	0.821

RQ 5.3. Table 6.7 shows that the CATM-based ranking models outperform the ranking models based on the basic time models and the ranking model that scores search results by the average values of the observed times between clicks. All improvements are statistically significant with $p < 0.05$. Thus, we conclude that the context-independent predictions of the CATMs trained on the time-between-clicks task provide a better means to rank search engine results than existing methods.

6.6 Related Work

We discuss two types of related work: behavioral signals used to improve and evaluate web search effectiveness (§6.6.1); and models of user behavior used to interpret these signals (§6.6.2).

6.6.1 Behavioral information

Modern search engines log a large number of user behavioral signals to improve and evaluate their effectiveness. We classify them in two groups: behavioral signals based on user actions and behavioral signals based on times between user actions.

Behavioral signals based on user actions. User clicks provide an important source of implicit user feedback [1–3, 26, 27, 29, 61, 80, 134, 141]. They have been used (i) to infer user preferences over search results, i.e., target values for learning a ranking function [27, 80], (ii) to design features for learning a ranking function [1, 2, 26] and for other applications [3, 61], and (iii) to compare ranking functions [29, 134, 141]. Some work distinguishes different types of clicks: first click [26, 141], last click [26], long dwell time click [26], satisfied click [141] and only click [26]. Next to clicks, some recent work considers mouse cursor movements on SERPs [39, 73, 74].

Behavioral signals based on times between user actions. Click dwell time, i.e., time spent by a user on the landing page of a search result, provides valuable information about user satisfaction with the clicked result [89]. Existing work uses it as a feature for ranking [1, 2, 26] and in many tasks related to user satisfaction [3, 61–63, 89, 107, 108, 125]. Times from a submission of a query to (i) the first click [24, 29, 61, 141], (ii) the first satisfied click [141], and (iii) the last click [29] are used as features to predict user satisfaction with the clicked result [61], compare two versions of a search engine [29, 141] and to cluster users based on their SERP examination strategies [24]. Song et al. [147] use the time since a user issued an abandoned query (i.e., a query for which there were no interactions with the search results) to the next query submission for classifying the abandoned query into positively abandoned and negatively abandoned. Dupret and Lalmas [44] use times between search engine visits to compare two versions of a search engine.

6.6.2 Models of user behavior

Now that we have described behavioral signals, we focus on their interpretation. To interpret a signal, we need to have a model that explains it. We start with models for explaining behavioral signals based on user actions and then discuss models for explaining behavioral signals based on times between user actions.

Modeling user actions. The simplest way to interpret click data is to compute *click-through rates* (CTRs), i.e., the ratios of the total number of clicks and the total number of impressions observed for a group of search engine results. Unfortunately, CTRs suffer from the so-called *position bias effect*, i.e., results presented at higher positions receive more clicks than results of similar quality presented at lower positions [36, 80, 81], which leads to suboptimal performance when using CTRs for ranking. To account for position bias, a large number of *click models* have been proposed [34].

Click models make a few assumptions about user interactions with search results, which ultimately allow them to obtain per query-result scores that show better correlation with relevance than CTRs. Among the most common assumptions are (i) the *linear traversal assumption* that a user scans search results from top to bottom [36]; and (ii) the *examination hypothesis* that a user clicks on a search result if, and only if, she examined the search result and is attracted by it [36]. Recently, it has been shown that patterns of user click behaviour can be learned automatically from interaction data [17]. In addition to position bias, recent work examines other types of bias including (i) *vertical bias* driven by visually salient vertical results (e.g., image results, video results, news results) [30, 33, 156]; (ii) *query bias*, which occurs if a query does not match the user's information need [179], (iii) *duplicate bias*, which occurs if a result has been examined earlier in the search task [179]; and (iv) bias driven by individual differences between users [144].

The notion of context bias, introduced in our work for times between user actions, generalizes the idea of bias in user actions (in particular, clicks and mouse hovers), and the proposed context-aware time model should be able to account for them with appropriate representations of the context.

Modeling times between user actions. The simplest way to interpret times between user actions is to compute their mean values. The average click dwell time is frequently used as a feature for ranking [1, 2, 26] and other applications [3, 61–63, 107, 108, 125]. The average time between a submission of a query to the first click is used both as a feature [61] and as an online metric for comparing two versions of a search engine [29, 141]. The average time between a submission of a query and (i) first click classified as satisfied, (ii) last click are also used as online metrics for comparing two versions of a search engine [29, 141].

A more sophisticated way to interpret times between user actions is to fit a probability distribution [89, 106]. Liu et al. [106] find that the Weibull distribution provides better goodness-of-fit to click dwell times than the exponential distribution. The authors provide an interesting interpretation for the shape parameter of the fitted Weibull distribution, which justifies the task of modeling the full probability distribution rather than just the mean of the distribution. Our work differs from their work in that we do not make the context-independence assumption that the observed times were drawn from the same probability distribution; we predict the probability distributions separately for each click using its context, and the information about click dwell times observed for the given query-result pair in other contexts. Liu et al. [106] also show that it is possible to predict parameters of the Weibull distribution for a given result using the information about the result's landing page, such as HTML tags, frequent words that occur on the page and times to download/render/parse the page. Our approach differs from their method in that we use contextual, i.e., result-independent, information; thus, our work is complementary to that of [106].

Kim et al. [89] fit gamma distributions to click dwell times, observed in a predefined click segment and labeled as satisfied or dissatisfied. The authors use the fitted distributions to classify new clicks into satisfied and dissatisfied. In particular, they compute the following features: (i) the differences between the fitted parameters of the satisfied and dissatisfied click dwell time distributions for the clicked result's segment, (ii) their expected values and the difference between them, (iii) the absolute differences between the observed click dwell time and the expected values of the satisfied and dissatisfied distributions, (iv) the log-likelihoods of the observed click dwell time according to the satisfied and dissatisfied distributions and the difference between them. The large number of features used in their work shows the advantage of modeling the full probability distribution over modeling just the mean. Similar features, computed from the probability distributions predicted in our work can potentially be used in a broad range of applications.

Our work is the first to systematically address the problem of modeling times between user actions. We introduce the notion of context bias effect and propose a context-aware time model that predicts times elapsed between user actions using both the ID of the first action (which is what existing methods rely on) and the context in which it takes place (our novelty).

6.7 Conclusions and Future Work

In this chapter, we answered the following research question:

RQ 5 How to correctly interpret times between user actions observed in different contexts?

We introduced the notion of *context bias effect* in times between user actions (i.e., a difference in probability distributions of times associated with the same user action, but observed in different contexts); and proposed a *context-aware time model* (CATM) that allows us to estimate parameters of a probability distribution of the time elapsed between user actions in a given context. CATM's ability to account for user context can be used to predict times between user actions in a context in which these actions have not previously been observed, and to obtain context-independent estimates of times between actions by predicting them in predefined contexts.

We showed that, for 37%–80% of query-result pairs (q, r), depending on the number of observations, the distributions of times elapsed between a click on the result r and the next click on the same SERP differed in sessions, in which the result r was the first item to be clicked, and in sessions, in which there were clicks on other results before the result r was clicked. Similarly, we showed that previous user interactions in a search task influence distributions of times between (i) submission of a query and the first click on a SERP, (ii) submission of a query and the last click on a SERP, and (iii) submission of an abandoned query (i.e., a query with no clicks on a SERP) and the next query submission.

We evaluated the context-aware time model on four temporal prediction tasks (i.e., to predict the time elapsed between a pair of user actions) and a ranking task (i.e., to rank a set of results by their relevance to a query). The results on the temporal prediction tasks show that the proposed context-aware time model, which makes use of both the ID of the first action and the context in which it takes place, provides a better means to explain times between user actions than existing methods, which rely solely on the first action ID. The results on the ranking task show that the context-independent estimates of times between consecutive clicks, obtained with the context-aware time model, allow us to construct rankings that result in better performance in terms of relevance than the rankings produced by the mean values of the observed times between consecutive clicks.

As to future work, we plan to consider other representations of the context. Besides using context-independent estimates of times between consecutive clicks as features for ranking, the predictions of context-aware time models can be used in a range of other applications that use the average or predicted times between user actions. These applications include prediction of click satisfaction [89], result usefulness [107], search task difficulty [3, 108], search goal success [61, 62], urgent information needs [119], struggling vs. exploring behavior [63, 125], positive vs. negative abandonment [147] and clustering users based on their SERP examination strategies [24].

7 Metafeatures

In the preceding research chapters, we represented queries and documents by their IDs. Using such representations allows to make accurate predictions for SERPs consisting of frequent query-document pairs, but does not generalize to new queries/documents/query-document pairs. Therefore, in this chapter, we move our focus to content-based models that represent queries and documents by their text. And in particular, to latent semantic models, which have been proposed to bridge the lexical gap between queries and documents that is due to the fact that searchers and content creators often use different vocabularies and language styles to express the same concept.

Modern search engines simply use the outputs of latent semantic models as features for a so-called global ranker. We argue that this is not optimal, because a single value output by a latent semantic model may be insufficient to describe all aspects of the model's prediction, and thus some information captured by the model is not used effectively by the search engine. Specifically, we answer the following research question asked in §1.1:

RQ 6 How to extract potentially useful information from a trained latent semantic model and how to utilize this information for improving ranking of search results?

To increase the effectiveness of latent semantic models in web search, we propose to create *metafeatures*—feature vectors that describe the structure of the model's prediction for a given query-document pair—and pass them to the global ranker along with the models' scores. We provide simple guidelines to represent the latent semantic model's prediction with more than a single number, and illustrate these guidelines using several latent semantic models.

We test the impact of the proposed metafeatures on a web document ranking task using four latent semantic models. Our experiments show that (i) through the use of metafeatures, the performance of each individual latent semantic model can be improved by 10.2% and 4.2% in NDCG scores at truncation levels 1 and 10; and (ii) through the use of metafeatures, the performance of a combination of latent semantic models can be improved by 7.6% and 3.8% in NDCG scores at truncation levels 1 and 10, respectively.

This chapter is based on Borisov, Serdyukov, and de Rijke [18].

7.1 Introduction

For the majority of cases in which search engine users complain that they cannot find information, while the information does exist in the system, the reasons are due to a mismatch between terms in queries and documents [100]. Term mismatch happens because searchers and content creators often use different vocabularies and language styles to refer to the same concepts [50]. To bridge this lexical gap between queries and documents, *latent semantic models* have been proposed [100].

Today, the use of latent semantic models by search engines is restricted to simply passing their outputs as features to a so-called global ranker, along with outputs of other models used for ranking. We argue that this is not optimal, because a single value output by a latent semantic model may be insufficient to describe all aspects of the latent semantic model's prediction. Let us illustrate this using the Latent Semantic Indexing model [37].

The score produced by the Latent Semantic Indexing model is a sum of scores per latent space dimension. This is where potentially useful information for a global ranker may go missing as the model may assign similar scores to documents with radically different sets of per dimension scores. See Figure 7.1 for an example of this phenomenon: the sums of the per dimension scores are the same, but the lack of information about the distributions of their values, illustrated by this example, may hinder the global ranker when it attempts to reliably rank the two documents.

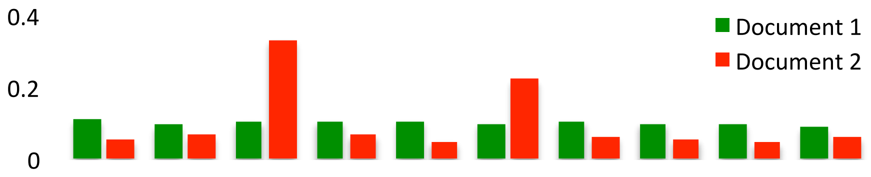


Figure 7.1: The plot shows the per dimension scores computed for two documents by the Latent Semantic Indexing model. The sums of the scores are the same, but the distributions of their values differ in interesting ways. (Best viewed in color.)

In previous work on latent semantic models in web search, comparisons of models are performed on a web document ranking task using the models' scoring functions [6, 10, 22, 51, 52, 75, 145, 160, 162, 167, 173]. While the scores produced by latent semantic models have demonstrated a strong correlation with document relevance, they are just the "tip of the iceberg" in capturing the relation between a query and document.

We argue that considering a latent semantic model's score only is not enough to determine its effectiveness in search, and all potentially useful information captured by the model should be considered. To increase the effectiveness of latent semantic models in search engines, we propose to expose the structure of their predictions to the global ranker. This is done through the use of *metafeatures*—feature vectors that we construct for a latent semantic model to describe its prediction and the way it arrived at this prediction. The latent semantic models' metafeatures are passed to the global ranker along with the models' scores.

The problem of creating metafeatures requires a solid understanding of the model's internal workings. However, as we will see, for many latent semantic models, their scoring functions give a good starting point. In particular, we provide two simple and broadly applicable guidelines for creating metafeatures based on the scoring functions used by latent semantic models, and demonstrate the effectiveness of these guidelines by inferring metafeatures from different latent semantic models in a systematic manner.

We test our ideas for complementing a latent semantic model's scores with metafeatures on a web document ranking task, and demonstrate that metafeatures provide a means to improve the performance of both (i) individual latent semantic models and (ii) combinations of multiple latent semantic models.

The proposed approach for creating and using metafeatures is substantially more than just feature engineering (where a significant part of the efforts of search engine employees goes). The process of *feature engineering* described by Domingos [41] consists of three steps: (i) to choose a particular machine learning algorithm and a target evaluation criterion (i.e., the difference in NDCG scores between the global ranker trained with and without candidate features); (ii) to generate a large number of candidate features; and (iii) to select the best features according to the chosen criterion. However, this approach is difficult to apply in practice, because there are no clear guidelines for selecting the candidate features (and, thus, the effectiveness of features often depends more on an individual's intuition rather than on a solid methodology). To alleviate this problem, we propose a *methodology* that guides the design choice of new features. Our proposal is both novel and ensures a broader impact of our technical contribution beyond the particular task setting (web search) that we consider.

We discuss related work in §7.2. Our method for creating metafeatures for latent semantic models and passing them along to a global ranker is detailed in §7.3. We detail our experimental setup in §7.4 and present our results and analysis in §7.5. We conclude in §7.6.

7.2 Related Work

We discuss three types of related work: latent semantic models, ranking models based on latent semantic models, and combinations of latent semantic models.

7.2.1 Latent semantic models

Latent Semantic Indexing (LSI) [37] uses singular value decomposition (SVD) of a document-term matrix to map queries and documents to low-dimensional concept vectors. The relevance of a document to a query is assumed to be proportional to the cosine similarity between their concept vectors. A major limitation of LSI that prevents it from being used in very large scale applications, is the computational cost of SVD. To overcome this limitation, a Regularized Latent Semantic Indexing (RLSI) [160] with an efficient implementation in MapReduce has been proposed.

Probabilistic Latent Semantic Indexing [70] views documents as mixtures of topics and ranks the documents by the probability of the query given the document distribution over topics. Latent Dirichlet Allocation (LDA) [13, 162] extends PLSI and assumes that topic distributions have a Dirichlet prior.

With increasingly large volumes of user logs, supervised latent semantic models trained on (clicked/not clicked) query-document pairs [6, 10, 22, 51, 52, 75, 145, 167, 173] have begun to outperform many unsupervised latent semantic models trained on documents only. Supervised Semantic Indexing (SSI) [6] and Regularized Mapping to Latent Spaces (RMLS) [167] learn weights for each query-document term pair using low rank matrix decomposition. The Bilingual Topic Model (BLTM) [52] is an extension of LDA, where queries and relevant documents are assumed to share the same distribution over topics, while they might use different topic word distributions for queries and documents.

Another approach to document ranking is based on statistical machine translation [10, 51]. The Word-based Translation Model (WTM) [10] ranks documents by the probability of the query given the document unigram translation model. The Phrase-based Translation Model (PTM) [51] extends WTM with phrases.

Recently, latent semantic models based on neural networks [22, 52, 75, 139, 145, 173] have gained popularity. The Discriminative Projection Model (DPM) [52, 173] maps queries and document to concept vectors with the siamese network architecture [22]. Salakhutdinov and Hinton [139] propose a deep generative model that maps documents to memory addresses in such a way that semantically similar documents are located at nearby addresses. The Deep Structured Semantic Model (DSSM) [75] uses a deep feed-forward neural network to map queries and documents to concept vectors. DSSM works at the level of character-trigrams, which allows generalization to unseen word forms. The performance of DSSM degrades as the text length increases, as its “bag of character-trigrams” representation leads to a combinatorial blow-up. The Convolutional Latent Semantic Model (CLSM) [145] mitigates this disadvantage by applying a DSSM-like model to word n-grams and combining their vectors at a later stage.

In our experiments we use LDA, WTM, DPM and DSSM as typical representatives of different families of models: topic models (PLSI, LDA, BLTM), translation models (WTM, PTM), conventional models based on matching in latent space (LSI, RLSI, SSI, RMLS, DPM) and recent models working at the level of character trigrams (DSSM, CLSM).

7.2.2 Ranking with latent semantic models

The training objective for LSI and RLSI is to minimize the reconstruction error of the document “bag of words” representation; PLSI and LDA (BLTM) minimize the perplexity of the document corpus (clicked query-document pairs); SSI, DPM minimize the ranking loss between clicked and unclicked documents. Common scoring functions used by the latent semantic models are (i) the cosine similarity between the query and document concept vectors (LSI, RLSI, SSI, RMLS, DPM, DSSM, CLSM) and (ii) the query likelihood function (PLSI, LDA, BLTM, WTM, PTM). These scoring functions are simple and intuitive, but we argue that they are not expressive enough to tune latent semantic models for relevance prediction and that they do not use all potentially useful information from the model.

7.2.3 Combinations of latent semantic models

Most work on latent semantic models in search does not address the problem of combining latent semantic models and only provides a comparison of latent semantic models with each other [6, 7, 10, 22, 51, 52, 75, 145, 160, 162, 173]. Some work also uses linear interpolation with traditional retrieval models based on lexical matching (VSM, TFIDF, BM25) for comparison [7, 167, 173]. Wu et al. [167] test the performance of their proposed RMLS model by comparing the performance of a global ranker with baseline ranking models used as features against the global ranker with the baseline ranking models and RMLS. We provide an experimental analysis of the contribution of metafeatures to the combination of latent semantic models.

What we add on top of the work mentioned above is the following. First, we show that the common ways of using latent semantic models in web search (i.e., (i) to rank web documents by scores of the cosine similarity and query likelihood functions, and (ii) to pass these scores as features to global ranker) are suboptimal. Second, we propose an approach that extracts more information from latent semantic models and leverages this information to improve ranking performance of both individual latent semantic models and their combinations.

7.3 Method

To increase the effectiveness of latent semantic models in (web) search engines, we propose to complement their outputs with *metafeatures*, feature vectors that describe the structure of the predictions of latent semantic models for a given query-document pair, and pass them to a global ranker together with the latent semantic models' scores (see Figure 7.2).

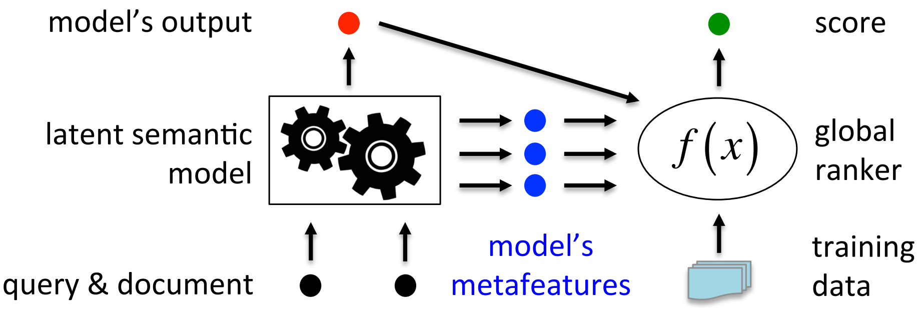


Figure 7.2: Adding metafeatures to the output of latent semantic models and passing their combination as features to a global ranker.

We start with a disclaimer. There is no general algorithm for generating metafeatures for an arbitrary latent semantic model. However, as we will see, for many latent semantic models, their scoring functions provide a good starting point. In §7.3.1, we offer two guidelines for creating metafeatures for a latent semantic model based on its scoring function. We operationalize these guidelines in §7.3.2 using a diverse set of latent semantic models.

7.3.1 Guidelines for creating metafeatures

We use Q to denote a query composed of terms q_1, q_2, …, $q_{|Q|}$ and D to denote a document composed of terms w_1, w_2, …, $w_{|D|}$.

We write $\mathrm{MF}_n(\mathrm{M}, P)$ to denote the n-th group of metafeatures for a latent semantic model or a class of such models, M; P denotes the parameters of the group of metafeatures. We use a compact form $\mathrm{MF}_{n_1,\ldots,n_k}(\mathrm{M}, P)$ to denote the metafeatures $\mathrm{MF}_{n_1}(\mathrm{M}, P)$, …, $\mathrm{MF}_{n_k}(\mathrm{M}, P)$.

Guideline 1 ("Divide and Conquer")

The values making up the score produced by a latent semantic model capture more potentially useful information than the latent semantic model's score by itself. They could, for instance, help to distinguish the two documents shown in Figure 7.1.

Below, we provide guidelines (a)–(c) for different types of scoring functions. We say that a scoring function $F(Q, D)$ can be expressed as a composition of functions $\mathcal{F} = \{f_i(Q, D)\}$, if for all query-document pairs (Q, D), it is possible to compute the value of $F(Q, D)$ by knowing the values of a subset of $\mathcal{F}$. The values of some $f_i(Q, D)$ might be not defined for a given query-document pair (Q, D), but it should still be possible to compute the value of $F(Q, D)$ using the values of other $f_i(Q, D) \in \mathcal{F}$. The *occurrence probability* of $f_i(\cdot)$ is the probability that $f_i(\cdot)$ is defined for a query-document pair (Q, D) randomly drawn from the (unknown) distribution of query-document pairs.

(a) For a scoring function $F(Q, D)$ that can be expressed as a composition of a fixed number (N) of functions $f_i(Q, D)$, define metafeatures $\mathrm{MF}_1(F)$ by composing them of the values of the functions $f_i(Q, D)$ for a query-document pair (Q, D):

$$\mathrm{MF}_1(F) \quad = \quad (f_1(Q, D), \ldots, f_N(Q, D)).$$

(b) For a scoring function $F(Q, D)$ that can be expressed as a composition of a very large or unlimited number of functions $f_i(Q, D)$, define metafeatures $\mathrm{MF}_1(F, N)$ of size N by composing them of the values of N functions $f_i(Q, D)$ that have the highest occurrence probabilities; if the value of function $f_i(Q, D)$ is not defined for a given query-document pair (Q, D), set the metafeatures' component that corresponds to the function $f_i(Q, D)$ with a NaN (Not a Number) value:

$$\mathrm{MF}_1(F, N) = \left(\widehat{f_1}(Q, D), \ldots, \widehat{f_N}(Q, D)\right),$$

where $\widehat{f_i}(Q, D)$ is $f_i(Q, D)$ if $f_i(Q, D)$ is defined, and NaN otherwise.

(c) For a scoring function $F(Q, D)$ that can be expressed as a composition of a very large or unlimited number of functions $f_i(Q, D)$ that have relatively high occurrence probabilities, define metafeatures $\mathrm{MF}_1(F, k, p_1, \ldots, p_N)$ by composing them of the descriptive statistics of the distribution of values of the functions $f_i(Q, D)$ for a given query-document pair (Q, D), e.g., the expected value $\mathbb{E}[f_i(Q, D)]$, variance $\mathrm{Var}[f_i(Q, D)]$, k min / max values of $f_i(Q, D)$ and percentiles p_i of $\{f_i(Q, D)\}$. The default value of k used in our study is 3; the default percentiles p_i are 0.01, 0.03, 0.05, 0.125, 0.25, 0.5, 0.75, 0.875, 0.95, 0.97, 0.99.

Note: For a scoring function $F(Q, D)$ that can be expressed as a composition of functions $f_i(Q, D)$ that come from two or more different groups $\mathcal{G} = \{g_k : k = 1, \ldots, |\mathcal{G}|\}$, define $|\mathcal{G}|$ scoring functions $F_k(Q, D)$ by setting the values of $f_i \notin g_k$ to constants, and construct metafeatures for $F_k(Q, D)$ using the guidelines (a)–(c).

Guideline 2 ("Find the Strongest and Weakest Links")

For some latent semantic models, there is only a limited number of ways to express a scoring function $F(Q, D)$ as a composition of functions $f_i(Q, D)$. E.g., the scoring functions of WTM and DSSM cannot be expressed as a composition of per document term functions $f_1(Q, D), \ldots, f_{|D|}(Q, D)$, where the functions $f_i(Q, D)$ describe all relevant information about the i-th terms in the document D for computing the scoring function $F(Q, D)$. But if such functions $f_1(Q, D), \ldots, f_{|D|}(Q, D)$ existed, their values could be regarded as contributions of the document terms to the scoring function $F(Q, D)$, and in this way, might help the global ranker to distinguish the documents that "answer" all query terms from those that match only some of them.

We suggest to create "surrogate functions" that perform the role of $f_i(Q, D)$. In particular, we propose to compute gradients of the scoring function $F(Q, D)$ with respect to the occurrences of document terms. This is a reasonable choice, because the components of these gradients measure how the score would change if we removed a small fraction of a term. Below, we capture this intuition more formally.

For a symmetric scoring function $F(Q, D)$ of query terms Q and document terms D (i.e., a function that takes the same value for any permutation of query terms and document terms), first define a function $F(\mathbf{q}, \mathbf{d})$ of vectors $\mathbf{q}$ and $\mathbf{d}$ of vocabulary size $|V|$, whose i-th components are the number of occurrences of the i-th term in query Q and document D, respectively. Then define a matrix M_Q of size $|Q| \times |V|$, whose i-th row is a one-hot vector with the component corresponding to the i-th query term q_i set to 1; and a matrix M_D of size $|D| \times |V|$, whose j-th row is a one-hot vector with the component corresponding to the j-th document term d_j set to 1. Finally, define the metafeatures $\mathrm{MF}_1(F, N_1)$ and $\mathrm{MF}_2(F, N_2)$ as gradients of $F(\mathbf{q}, \mathbf{d})$ with respect to $\mathbf{q}$ and $\mathbf{d}$, $\nabla_Q F(\mathbf{q}, \mathbf{d})$ and $\nabla_D F(\mathbf{q}, \mathbf{d})$, multiplied by the matrices $M_Q(N_1)$ and $M_Q(N_2)$ composed of the first N_1 and N_2 rows of the matrices M_Q and M_D:

$$\begin{aligned}\mathrm{MF}_1(F, N_1) &= M_Q(N_1)\nabla_Q F(\mathbf{q}, \mathbf{d}),\\ \mathrm{MF}_2(F, N_2) &= M_D(N_2)\nabla_D F(\mathbf{q}, \mathbf{d}).\end{aligned}$$

Sometimes, it is more convenient to apply a monotonic function $g(x)$ to the scoring function $F(\cdot)$, and use the resulting complex function $g(F(\cdot))$ instead of $F(\cdot)$.

As we show in the next section, the presented guidelines help to extract a lot of potentially useful metafeatures from existing state-of-the-art latent semantic models.

7.3.2 Application of our guidelines

In this section we demonstrate how to apply the guidelines presented in §7.3.1 to different families of latent semantic models. Note that we do not aim to extract all possible metafeatures that could be extracted according to Guidelines 1 and 2, as the main purpose of our study is not an exhaustive search for all of them (which could potentially be a very large number), but a demonstration of the benefits of the methodology of metafeature extraction in general. Thus, we focus on the extraction of the most promising and interpretable metafeatures in this section.

Latent semantic models based on the language modeling approach

We describe the metafeatures that we infer for topic models [13, 52, 70] and the Word-based Translation Model [10] that employ the language modeling approach. We start with the common metafeatures and then take a closer look at each model's scoring function.

Latent semantic models based on the language modeling approach score a query-document pair (Q, D) by the probability of the query Q given the document model M_D. Many latent semantic models make the "bag of words" assumption, which allows one to decompose the probability of the query into the product of the query term probabilities. Thus, the scoring function used by these models is the product of the query term probabilities given the document model, $P(q_i \mid M_D)$:

$$P(Q \mid M_D) = \prod_{i=1}^{|Q|} P(q_i \mid M_D). \tag{7.1}$$

Following Guideline 1 (b), we use the factors under the product sign of (7.1) to define metafeatures $\mathrm{MF}_1(\mathrm{QL}, N)$ that capture probabilities of the first N query terms given the document model:

$$\mathrm{MF}_1(\mathrm{QL}, N) \quad = \quad (P(q_1 \mid M_D), \ldots, P(q_N \mid M_D)). \tag{7.2}$$

These metafeatures may help the global ranker to distinguish between two documents that get very similar scores by the query likelihood scoring function, but for very different reasons. For example, consider two documents D_1 and D_2, such that:

- given the document model for D_1, all query terms $q_1, \ldots, q_{|Q|}$ have roughly the same probabilities; and
- given the document model for D_2, the first query term q_1 has a very low probability, and the other query terms $q_2, \ldots, q_{|Q|}$ have higher probabilities than given the document model for D_1.

Intuitively, these are two different cases. We want the global ranker to know about this through the use of metafeatures. The components of $\mathrm{MF}_1(QL, N)$ can be seen as query term contributions, and are in fact obtained using Guideline 2 for $g(x) = \log x$.

Some work [10, 51, 52] reports that $P(q_i \mid M_D)$ may be too coarse to be used for retrieval and suggests to use linear interpolation with the document unigram language model $P_{\mathrm{LM}}(q_i|D)$ [10], the collection unigram language model $P_{\mathrm{LM}}(q_i \mid C)$ [51] or both [52]:

$$\widetilde{P}(Q \mid M_D) = \prod_{i=1}^{|Q|} \left(\alpha_1 P_{\mathrm{LM}}(q_i \mid C) + \alpha_2 P_{\mathrm{LM}}(q_i \mid D) + \alpha_3 P(q_i \mid M_D)\right),$$

where $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Similarly, following Guideline 1 (b), we define metafeatures $\mathrm{MF}_{2,3}(\mathrm{QL}, N)$ that capture the probabilities of the first N query terms given the document language model and the collection language model:

$$\begin{aligned}\mathrm{MF}_2(\mathrm{QL}, N) &= (P_{\mathrm{LM}}(q_1 \mid C), \ldots, P_{\mathrm{LM}}(q_N \mid C))\,,\\ \mathrm{MF}_3(\mathrm{QL}, N) &= (P_{\mathrm{LM}}(q_1 \mid D), \ldots, P_{\mathrm{LM}}(q_N \mid D))\,.\end{aligned}$$

These metafeatures provide extra information about the individual query terms that might help the global ranker to make better use of $\mathrm{MF}_3(\mathrm{QL}, N)$. E.g., low query term probabilities given the latent semantic model $P(q_i \mid M_D)$ are less severe for query terms q_i for which $P_{\mathrm{LM}}(q_i \mid C)$ is high than for query terms q_i for which $P(q_i \mid M_D)$ is low. A high query term probability given by the document model, $P_{\mathrm{LM}}(q_i \mid D)$, indicates that a high probability of a query term given the latent semantic model, $P(q_i \mid M_D)$, is due to a lexical match. In this case, term-based models' predictions might be more reliable for the global ranker.

Topic models. Topic models (e.g., PLSI, LDA, BLTM) view documents as mixtures of topics, i.e., $M_D = \{P(t_j \mid D) : 1 \leq j \leq |T|\}$, where $P(t_j \mid D)$ denotes the probability of topic t_j given the document D, and $|T|$ denotes the number of topics. The probability of a query term q_i given document model M_D, $P(q_i \mid M_D)$, is defined as the sum of the query term probabilities given the topic, $P(q_i \mid t_j)$, weighted by topic probabilities, $P(t_j \mid D)$:

$$P(q_i \mid M_D) = \sum_{j=1}^{|T|} P(q_i \mid t_j) P(t_j \mid D).$$

The document score is a function of (i) topic probabilities given the document, $P(t_j \mid D)$ and (ii) query term probabilities given the topic, $P(q_i \mid t_i)$. Guideline 1 suggests that we define metafeatures $\mathrm{MF}_{1,2}(\mathrm{TM})$ that describe these values:

$$\begin{aligned}\mathrm{MF}_1(\mathrm{TM}) &= \big(P(t_1 \mid D), \ldots, P(t_{|T|} \mid D)\big)\,,\\ \mathrm{MF}_2(\mathrm{TM}, N) &= \begin{pmatrix} P(q_1 \mid t_1), & \ldots, & P(q_1 \mid t_{|T|}), \\ \ldots & \ldots, & \ldots \\ P(q_N \mid t_1), & \ldots, & P(q_N \mid t_{|T|}) \end{pmatrix}.\end{aligned}$$

However, the size of $\mathrm{MF}_2(\mathrm{TM}, N)$ might be too large to train the global ranker without causing overfitting. In essence, $\mathrm{MF}_1(\mathrm{TM})$ and $\mathrm{MF}_2(\mathrm{TM}, N)$ are meant to inform the global ranker about how well the document topic distribution helps to maximize the query probability. This can be achieved by comparing the document topic distribution with the query topic distribution—the query topic distribution is, by definition, the topic distribution that maximizes the query probability. This idea was originally proposed in [98], where the authors used Kullback-Leibler divergence between the query and document topic distributions as a scoring function:

$$KL(Q, D) = \sum_{i=1}^{|T|} P(t_i \mid Q) \log \frac{P(t_i \mid Q)}{P(t_i \mid D)}. \tag{7.3}$$

Based on preliminary experiments, the Kullback-Leibler divergence scoring function (7.3) performed much poorer than the Query Likelihood scoring function (7.1), but it happened to be useful for creating metafeatures. Following Guideline 1 (a), we define metafeatures $\mathrm{MF}_1(\mathrm{TM})$ (defined earlier) and $\mathrm{MF}_3(\mathrm{TM})$ that capture the query and document topic distributions:

$$\mathrm{MF}_3(\mathrm{TM}) \quad = \quad \big(P(t_1 \mid Q), \ldots, P(t_{|T|} \mid Q)\big)\,.$$

The Word-based Translation Model. The Word-based Translation Model casts document ranking as a statistical machine translation problem, in which query Q is assumed to be a translation of document D. The document model M_D is defined as a probability distribution of terms given a document. The probability of term w given document model M_D is defined as the average translation probability of document terms $w_1, \ldots,$ $w_{|D|}$ into term w. So the probability of query term q_i given document model M_D is:

$$P(q_i \mid M_D) = \frac{1}{|D|} \sum_{j=1}^{|D|} P(q_i \mid w_j).$$

The document score is a function of the translation probabilities of a document term into a query term. Following Guideline 1 (b) we define metafeatures $\mathrm{MF}_1(\mathrm{WTM}, N_1, N_2)$ that capture probabilities of the query terms given the document terms:

$$\mathrm{MF}_1(\mathrm{WTM}, N_1, N_2) = \begin{pmatrix} P(q_1 \mid w_1), & \ldots, & P(q_1 \mid w_{N_2}), \\ \ldots & \ldots, & \ldots \\ P(q_{N_1} \mid w_1), & \ldots, & P(q_{N_q} \mid w_{N_2}) \end{pmatrix}.$$

As the number of $P(q_i, w_j)$ that occur with a not low frequency is large, we also define metafeatures $\mathrm{MF}_2(\mathrm{WTM})$ from the set of translation probabilities of document terms into query terms $\{P(q_i, w_j)\}_{1 \leq i \leq |Q|, 1 \leq j \leq |D|}$, following Guideline 1 (c).

Guideline 2 suggests that passing on information about the query and document term contributions to the document score might be useful for the global ranker. The metafeatures $\mathrm{MF}_1(\mathrm{QL}, N)$ that we have defined earlier (see (7.2)) describe the query term contributions. To define metafeatures that describe the document term contributions, we first apply a monotonic function $g(x) = \log x$ to the scoring function $P(Q \mid M_D)$ that gives a new scoring function

$$F(Q, D) = g(P(Q \mid M_D)) = \sum_{i=1}^{|Q|} \log \frac{1}{|D|} \sum_{j=1}^{|D|} P(q_i \mid w_j).$$

Then we define a function $F(\mathbf{q}, \mathbf{d})$ of $\mathbf{q}$ and $\mathbf{d}$:

$$F(\mathbf{q}, \mathbf{d}) = \mathbf{q}^T (\log A\mathbf{d} - \log B\mathbf{d}),$$

where A and B are matrices of size $|V| \times |V|$ with $A_{i,j} = P(t_i, t_j)$, and $B_{i,j} = 1$; $\log(\mathbf{x})$ denotes the element-wise logarithm of vector $\mathbf{x}$. Finally, we define the metafeatures that capture quantitative contributions of document terms as follows:

$$\begin{aligned} \mathrm{MF}_3(F, N_2) \quad &= \quad M_Q(N_2) \nabla_D F(\mathbf{q}, \mathbf{d}) \\ &= \quad M_Q(N_2)(A^T \circ \mathrm{inv}(A\mathbf{d}) - B^T \circ \mathrm{inv}(B\mathbf{d}))\mathbf{q}, \end{aligned}$$

where $Z = X \circ \mathbf{y}$ denotes the element-wise multiplication of matrix X by vector $\mathbf{y}$: $Z[i,j] = X[i,j]\mathbf{y}[i]$; and $\mathbf{z} = \mathrm{inv}(\mathbf{x})$ denotes the element-wise multiplicative inverse of vector $\mathbf{x}$: $\mathbf{z}[i] = 1/\mathbf{x}[i]$, if $\mathbf{x}[i] \neq 0$ and 0 otherwise. Briefly, $\mathrm{MF}_3(\mathrm{WTM}, N_2)$ are:

$$\left(\sum_{i=1}^{|Q|} \frac{P(q_i \mid w_1)}{\sum_{j=1}^{|D|} P(q_i \mid w_j)} - \frac{|Q|}{|D|}, \dots, \sum_{i=1}^{|Q|} \frac{P(q_i \mid w_{N_2})}{\sum_{j=1}^{|D|} P(q_i \mid w_j)} - \frac{|Q|}{|D|}\right).$$

The components of $\mathrm{MF}_3(\mathrm{WTM}, N_2)$ characterize the contributions of document terms w_i with respect to the other terms in the document D: $\alpha_k = \sum_{i=1}^{|Q|} \frac{P(q_i|w_k)}{\sum_{j=1}^{|D|} P(q_i|w_j)}$ is the actual quantitative contribution of the document term w_i; $\beta = \frac{|Q|}{|D|} = \frac{1}{|D|}\sum_{k=1}^{|D|} \alpha_k$ is the average contribution of the document terms $w_1, \dots, w_{|D|}$.

Latent semantic models based on the latent space matching approach

Next, we describe metafeatures that we infer for latent semantic model that employ the latent space matching approach. We start with metafeatures shared by all models of this class and then take a closer look at the Deep Structured Semantic Model [75].

Latent semantic models based on the latent space matching approach learn vector representations for queries and documents, such that the distance between a query vector v_Q and a document vector v_D reflects the degree of relevance of the document D to the query Q. The standard choice for a distance function between the query and document vectors is the cosine similarity measure [112]:

$$\cos(v_Q, v_D) = \frac{v_Q \cdot v_D}{\mid v_Q \mid\mid v_D \mid} = \sum_{i=1}^{N} \frac{v_Q^{(i)} v_D^{(i)}}{\mid v_Q \mid\mid v_D \mid}, \tag{7.4}$$

where N denotes the dimensionality of the vector space and $v^{(i)}$ denotes the i-th component of vector v.

Following Guideline 1 (a), we use the terms under the summation sign in (7.4) to define metafeatures $\mathrm{MF}_1(\mathrm{COS})$ that capture the per dimension matching scores:

$$\mathrm{MF}_1(\mathrm{COS}) = \left(\frac{v_Q^{(1)} v_D^{(1)}}{\mid v_Q \mid\mid v_D \mid}, \dots, \frac{v_Q^{(N)} v_D^{(N)}}{\mid v_Q \mid\mid v_D \mid}\right).$$

The rationale is analogous to the one underlying $\mathrm{MF}_1(\mathrm{QL}, N)$: we want to be able to distinguish the two cases shown in Figure 7.1. One document has roughly the same scores in all dimensions, the other one has much higher scores in two dimensions and lower scores in other dimensions.

Viewed differently, the document score is a function of both the query vector v_Q and the document vector v_D. Hence, following Guideline 1 (a), we define metafeatures $\mathrm{MF}_{2,3}(\mathrm{COS})$ that capture the components of the query vector v_Q and document vector v_D:

$$\begin{aligned}\mathrm{MF}_2(\mathrm{COS}) &= \left(v_Q^{(1)}, \dots, v_Q^{(N)}\right),\\ \mathrm{MF}_3(\mathrm{COS}) &= \left(v_D^{(1)}, \dots, v_D^{(N)}\right).\end{aligned}$$

The Deep Structured Semantic Model. The Deep Structured Semantic Model (DSSM) uses a technique called *word hashing*, which represents queries and documents with letter-trigram vectors. For example, *San Francisco* is encoded with the following letter-trigrams #sa, san, an#, #fr, fra, ran, anc, nci, cis, isc, sco, co#, where # is a boundary symbol. These letter-trigram vectors are passed through the three fully connected layers of a feed-forward neural network to obtain vector representations v_Q and v_D, as illustrated in Figure 7.3.

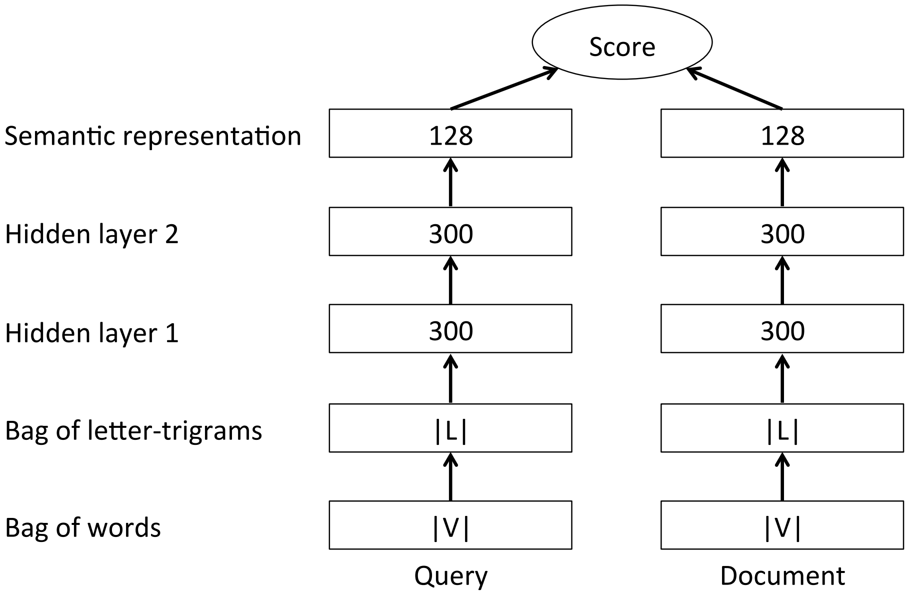


Figure 7.3: The Deep Structured Semantic Model (DSSM) [75].

Guideline 2 suggests that we construct metafeatures that capture the quantitative contributions of query and document terms to the document score. But since DSSM works at the letter-trigram level, we consider letter-trigram contributions.[1]

We use the following notation: $|T|_c$ is the length of text T in characters; $|L|$ is the total number of letter-ngrams; $\mathbf{q_c}$ and $\mathbf{d_c}$ are vectors of size $|L|$, whose elements on the i-th position are the numbers of occurrences of the i-th letter-ngram in query Q and document D, respectively; $v_Q(\mathbf{q_c})$ and $v_D(\mathbf{d_c})$ are functions of $\mathbf{q_c}$ and $\mathbf{d_c}$ that return the semantic representations of query Q and document D (these functions are learned by DSSM); M_Q is a matrix of size $|Q|_c \times |V|$, whose i-th row is a one-hot vector with the component corresponding to the i-th query letter-trigram set to 1; M_D is a matrix of size $|D|_c \times |V|$, whose i-th row is a one-hot vector with the component corresponding to the i-th document letter-trigram set to 1. We write $M_Q(N_1)$ and $M_D(N_2)$ for the matrices composed of the first N_1 and N_2 rows of the matrices M_Q and M_D.

We define metafeatures that account for the contributions of the query and document

[1]In the case of the DSSM model, term contributions are the sums of the contributions of the letter-trigrams that constitute the term.

letter-trigrams as follows:

$$\begin{aligned}\mathrm{MF}_1(\mathrm{DSSM}, N_1) &= M_Q(N_1)\nabla_{\mathbf{q_c}} \cos\left(v_Q(\mathbf{q_c}), v_D(\mathbf{d_c})\right), \\ \mathrm{MF}_2(\mathrm{DSSM}, N_2) &= M_Q(N_2)\nabla_{\mathbf{d_c}} \cos\left(v_Q(\mathbf{q_c}), v_D(\mathbf{d_c})\right).\end{aligned}$$

Using the chain rule $\nabla_{\mathbf{x}} F(\mathbf{y}(\mathbf{x})) = J^T_{\mathbf{x}}\left(\mathbf{y}(\mathbf{x})\right) \nabla_{\mathbf{y}} F(\mathbf{y})$, where $J_{\mathbf{x}}\left(\mathbf{y}(\mathbf{x})\right)$ denotes the Jacobian matrix of a vector function $\mathbf{y}(\mathbf{x})$ of a vector $\mathbf{x}$ w.r.t. the vector $\mathbf{x}$, and the equality $\nabla_{\mathbf{x}} \cos(\mathbf{x}, \mathbf{y}) = \nabla_{\mathbf{x}} \frac{\mathbf{x}^T \mathbf{y}}{|\mathbf{x}|\,|\mathbf{y}|} = \frac{\mathbf{y}}{|\mathbf{x}|\,|\mathbf{y}|} - \frac{\mathbf{x}^T \mathbf{y}}{|\mathbf{x}|^3 |\mathbf{y}|}$, we rewrite $\mathrm{MF}_{1,2}(\mathrm{DSSM}, N_1)$ as follows:

$$\begin{aligned}\mathrm{MF}_1(\mathrm{DSSM}, N_1) &= M_Q(N_1) J^T_{\mathbf{q_c}}(v_Q(\mathbf{q_c})) \nabla_{v_Q} \cos\left(v_Q, v_D\right) \\ &= M_Q(N_1) J^T_{\mathbf{q_c}}(v_Q(\mathbf{q_c})) \left(\frac{v_Q}{\mid v_Q \mid\mid v_D \mid} - \frac{v_D^T v_Q}{\mid v_Q \mid^3 \mid v_D \mid} \right), \\ \mathrm{MF}_2(\mathrm{DSSM}, N_2) &= M_Q(N_2) J^T_{\mathbf{d_c}}(v_D(\mathbf{d_c})) \nabla_{v_D} \cos\left(v_Q, v_D\right) \\ &= M_Q(N_2) J^T_{\mathbf{q_c}}(v_D(\mathbf{d_c})) \left(\frac{v_D}{\mid v_Q \mid\mid v_D \mid} - \frac{v_D^T v_Q}{\mid v_Q \mid\mid v_D \mid^3} \right).\end{aligned}$$

The elements of $J_{\mathbf{q_c}}(v_Q(\mathbf{q_c}))$ and $J_{\mathbf{d_c}}(v_D(\mathbf{d_c}))$ are the first derivatives of the functions in the output layer of the neural network w.r.t. the input vectors $\mathbf{q_c}$ and $\mathbf{d_c}$ (see e.g., [8] for the exact formulas).[2]

The metafeatures $\mathrm{MF}_1(\mathrm{DSSM}, N_1)$ and $\mathrm{MF}_2(\mathrm{DSSM}, N_2)$ do not capture information about word boundaries. This might be a disadvantage because the importance of a letter-trigram, which we define as its expected contribution, varies with its position. E.g., letter-trigrams at the end of a term are typically less informative than those in the middle (for many European languages) [71].

To capture the boundaries between terms in the query and terms in the documents we define metafeatures $\mathrm{MF}_{3,4,5,6}(\mathrm{DSSM}, N)$:

$$\begin{aligned}\mathrm{MF}_3(\mathrm{DSSM}, N_1) &= \left(start(Q, 1), \ldots, start(Q, N_1)\right), \\ \mathrm{MF}_4(\mathrm{DSSM}, N_1) &= \left(end(Q, 1), \ldots, end(Q, N_1)\right), \\ \mathrm{MF}_5(\mathrm{DSSM}, N_2) &= \left(start(D, 1), \ldots, start(D, N_2)\right), \\ \mathrm{MF}_6(\mathrm{DSSM}, N_2) &= \left(end(D, 1), \ldots, end(D, N_2)\right),\end{aligned}$$

where $start(T, p)$ and $end(T, p)$ denote the offsets of the p-th letter-trigram in the text T from the first and the last trigrams of the term it falls. An illustration is given in Figure 7.4.

Note that for more sophisticated latent semantic models, the metafeatures can be more sophisticated. E.g., following Guideline 2 for DSSM and WTM we arrive at the most non-trivial metafeatures among the ones that we have proposed, such as $\mathrm{MF}_1(\mathrm{DSSM}, \mathrm{N}_1)$, $\mathrm{MF}_2(\mathrm{DSSM}, \mathrm{N}_2)$ and $\mathrm{MF}_3(\mathrm{WTM}, \mathrm{N}_2)$. However, as we demonstrate in our experiments below, all metafeatures, regardless of their complexity, are useful for improving retrieval quality.

[2] Since DSSM uses weight sharing, $J_{\mathbf{q_c}}(v_Q(\mathbf{q_c})) = J_{\mathbf{d_c}}(v_D(\mathbf{d_c}))$.

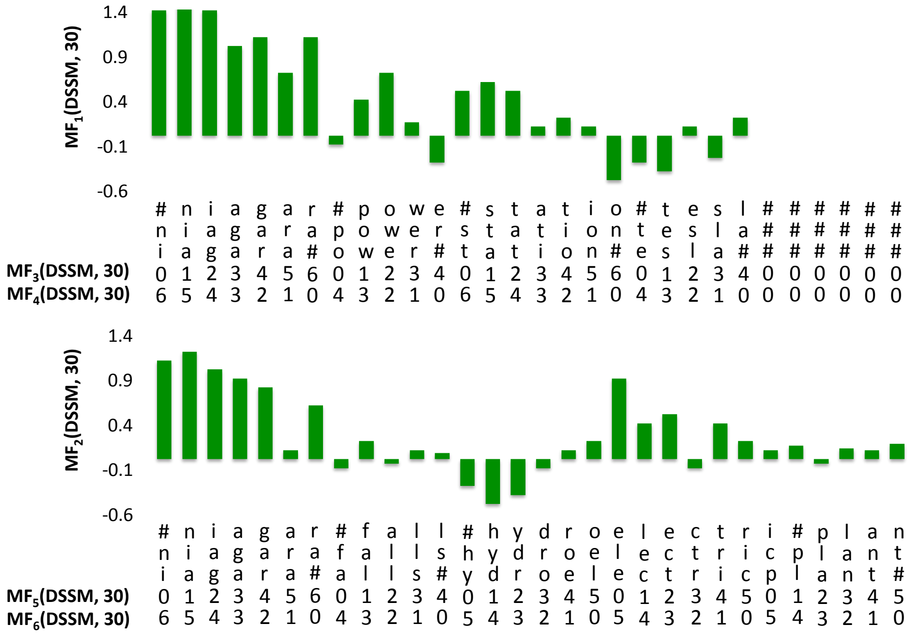


Figure 7.4: Metafeatures $\mathrm{MF}_{1,2,3,4,5,6}(\mathrm{DSSM}, 30)$ for query *"niagara power station tesla"* and document *"niagara falls hydroelectric plant"*.

7.4 Experimental Setup

7.4.1 Research questions

We split RQ 6 into two subquestions:

RQ 6.1 Do metafeatures associated with latent semantic models help improve the performance of individual latent semantic models on a web document ranking task? In particular,

- **(a)** Which of the metafeatures defined in §7.3.2 help to improve performance of the underlying latent semantic models?
- **(b)** Does employing a combination of different metafeatures inferred from a single latent semantic model yield better performance than individual metafeatures?
- **(c)** How does the performance of the global ranker, trained with metafeatures, vary with the size of the training data set?

RQ 6.2 Do metafeatures associated with latent semantic models provide a means to improve the performance of a combination of latent semantic models on the web document ranking task?

7.4.2 Data sets and evaluation methodology

In modern search engines, a document is described by multiple fields, including body text, title, URL and anchor texts [149]. In our experiments, we focus on the title field. We do this for the following reasons. First, recent work on supervised latent semantic models in web search focuses on the title field [51, 52, 75, 145, 167].[3] Second, the title field gives better [145] or, at least, as effective [149] retrieval results as the body text field (using BM25). Third, some models achieve state-of-the-art performance using the title field, but do not apply to other fields (e.g., it is not feasible to train DSSM [75] on the body text field, because of the extremely large number of non-zero elements in the word hashing layer).

To train latent semantic models, we collected a training data set that comprises 130,024,971 search sessions sampled from a commercial web search engine over a six months period.[4] It contains a total of 51,117,758 unique user queries and 178,289,551 retrieved documents. Figures 7.5 and 7.6 show the query and document distributions of the number of words and characters in the training data set. We use these statistics to set the parameters of metafeatures in our experiments (§7.4.4).

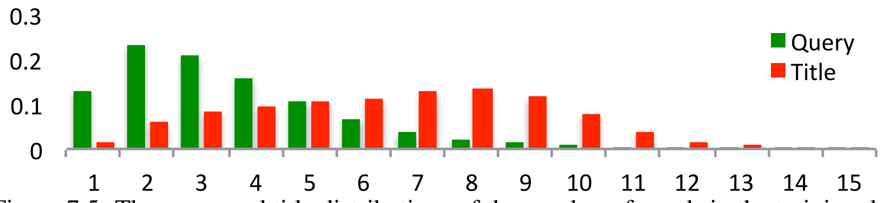


Figure 7.5: The query and title distributions of the number of words in the training data set. (Best viewed in color.)

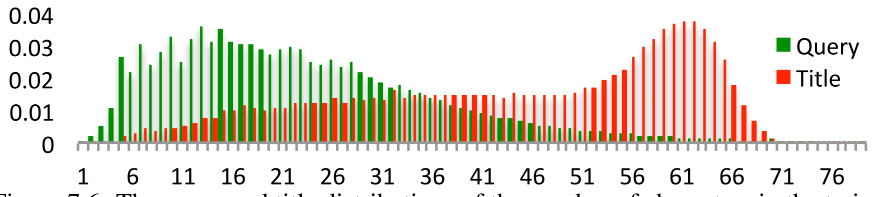


Figure 7.6: The query and title distributions of the number of characters in the training data set. (Best viewed in color.)

To evaluate the performance of the models, we collected an evaluation data set that contains 111,203 queries sampled from the query-log files of a commercial web search engine.[5] On average, each query is associated with 27 documents (URLs). Each query-document pair has a human-generated relevance label on a scale from 0 to 4, with 0 = bad, 1 = fair, 2 = good, 3 = excellent, 4 = perfect. Performance is measured

[3]Although our work targets supervised latent semantic models, we include a comparison with LDA, as a well-known baseline.

[4]There is no publicly available data set for training supervised latent semantic models [6, 10, 51, 52, 75, 145, 167, 173].

[5]There is no sufficiently large publicly available dataset with relevance labels and disclosed query and document terms (the TREC data set is too small to train the global ranker; the well-known Yahoo L2R and MSLR data sets do not contain query terms and document terms).

using mean Normalized Discounted Cumulative Gain (NDCG) [77] at truncation levels 1, 3, 5 and 10, using k-fold cross-validation. We use $(k-1)$ folds to train the global ranker and 1 fold to evaluate the performance; the results over k folds are averaged. We perform significance testing using a paired t-test; differences are considered statistically significant for p-values lower than 0.001.

7.4.3 Model settings and baseline performance

We use four latent semantic models for which we infer metafeatures. The baseline performance achieved by the models after optimization is listed in Table 7.1. Below we describe how the models are trained. For comparison, we also list a state-of-the art ranking model based on lexical matching, BM25.

BM25 (row 1 in Table 7.1) is a state-of-the-art document ranking model based on lexical matching. The parameters k_1 and b are optimized using Powell's method [132].

LDA (rows 2–3 in Table 7.1) is our MapReduce implementation of the topic model proposed in [13]. It is trained with 200 iterations of Gibbs sampling using $\alpha = 50/|T|$ and $\beta = 0.01$ (the default values used, e.g., in [162]). The number of topics $|T|$ is set to 100 (used, e.g., in [52]).[6] We consider ranking models based on the unsmoothed LDA model and the LDA model smoothed using a title unigram language model and a background unigram language model. The ranking model based on the smoothed version of LDA model (row 3) outperforms the ranking model based on the unsmoothed version of LDA model (row 2) by a large margin. In the rest of our experiments we use the smoothed version.

WTM (rows 4–5 in Table 7.1) is our implementation of the Word-based Translation Model [10]. It is trained on the query-title pairs. We consider ranking models based on the unsmoothed WTM and the WTM smoothed using a title unigram language model and a background unigram language model. We find that the ranking model based on the smoothed version of WTM (row 5) outperforms the ranking model based on the unsmoothed version of WTM (row 4) by a large margin. In the rest of our experiments we use the smoothed version of WTM.

DPM (row 6 in Table 7.1) is our implementation of the Discriminative Projection Model proposed in [52, 173]. It is trained on clicked query-title pairs. The number of dimensions is set to 300 (the performance does not improve much for larger numbers [173], while the training time is linear in the number of dimensions).

DSSM (row 7 in Table 7.1) is our implementation of the Deep Structured Semantic Model proposed in [75]. It is trained by maximizing the conditional likelihood of the clicked query-title pairs. We use the configuration proposed in [75] (the numbers of neurons in the hidden layers are {300, 300, 128}).

ALL (row 8–9 in Table 7.1) is a combination of LDA (w/ smoothing), WTM (w/ smoothing), DPM and DSSM by a linear model (row 8) and Gradient Boosted Regression Trees (GBRT [49], row 9).

[6] Shen et al. [145] used LDA with 100 and 500 topics, but the observed changes were not marked as statistically significant.

All ranking models based on latent semantic models outperform the BM25 ranking model by a large margin, on all metrics. DSSM is the best performing individual latent semantic model, again on all metrics. The two combination models outperform all individual models and GBRT outperforms the linear model. In the rest of the experiments, we use GBRT as our global ranker.

Table 7.1: Performance of the baseline ranking models.

		NDCG			
#	Ranking model	@1	@3	@5	@10
1	BM25	0.520	0.583	0.631	0.709
2	LDA (w/o smoothing)	0.538	0.608	0.657	0.734
3	LDA (w/ smoothing)	0.552	0.619	0.667	0.741
4	WTM (w/o smoothing)	0.545	0.611	0.659	0.736
5	WTM (w/ smoothing)	0.560	0.626	0.673	0.746
6	DPM	0.537	0.602	0.648	0.724
7	DSSM	0.568	0.640	0.687	0.759
8	ALL (linear model)	0.580	0.649	0.694	0.764
9	ALL (GBRT)	0.604	0.664	0.706	0.772

7.4.4 Experiments

We design two experiments to answer our research questions.

Experiment 1. To answer RQ 6.1, we compare the performance of each individual latent semantic model's score against the output of the global ranker trained with the latent semantic model's score as well as the metafeatures inferred from the latent semantic model used as features.

In §7.3.2 we have introduced some metafeatures together (e.g., $\mathrm{MF}_2(\mathrm{COS})$ and $\mathrm{MF}_3(\mathrm{COS})$) and provided the intuitions why those metafeatures might be helpful for the global ranker. We evaluate the impact of the metafeatures within the groups where they were introduced (Table 7.2).

For the metafeatures $\mathrm{MF}_{1,2,3}(\mathrm{QL}, N)$, $\mathrm{MF}_1(\mathrm{WTM}, N_1, N_2)$ and $\mathrm{MF}_2(\mathrm{WTM}, N)$ we set $N = N_1 = N_2 = 15$, because less than 0.3% of the queries and 0.2% of the titles in the training set contain more than 15 words (Figure 7.5). For the metafeatures $\mathrm{MF}_{1,2,3,4,5,6}(\mathrm{DSSM}, N)$ we set $N = 80$ because less than 0.4% of the queries and 0.2% of the titles in the training set contain more than 80 characters (Figure 7.6).

Experiment 2. To answer RQ 6.2, we compare the performance of the global ranker trained with the individual latent semantic models' scores only (for LDA, WTM, DPM, DSSM) against the global ranker trained with those individual latent semantic models' scores plus the metafeatures inferred from those models.

Table 7.2: Experiment 1 (runs).

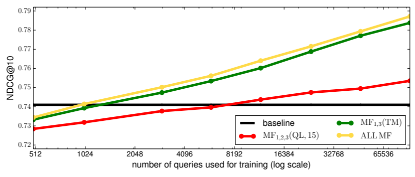
Metafeatures	Latent semantic models
$\mathrm{MF}_{1,2,3}(\mathrm{QL}, 15)$	LDA, WTM
$\mathrm{MF}_{1,3}(\mathrm{TM})$	LDA
$\mathrm{MF}_{1}(\mathrm{WTM}, 15, 15)$	WTM
$\mathrm{MF}_{2}(\mathrm{WTM})$	WTM
$\mathrm{MF}_{1}(\mathrm{QL}, 15), \mathrm{MF}_{3}(\mathrm{WTM}, 15)$	WTM
$\mathrm{MF}_{1}(\mathrm{COS})$	DPM, DSSM
$\mathrm{MF}_{2,3}(\mathrm{COS})$	DPM, DSSM
$\mathrm{MF}_{1,2,3,4,5,6}(\mathrm{DSSM}, 80)$	DSSM

7.5 Results

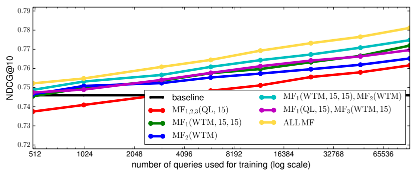
We present the outcomes of the two experiments described in §7.4.4, and provide answers to our research questions.

7.5.1 Experiment 1

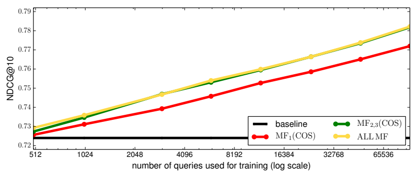
The results of Experiment 1 are given in Table 7.3 and Figure 7.7. Table 7.3 compares the performance of each individual latent semantic model and the global ranker trained using the latent semantic model's score together with the metafeatures listed in Table 7.2 as features. Figure 7.7 plots the performance (in terms of NDCG@10) for the latent semantic models with the metafeatures listed in Table 7.2 vs. the number of queries used for training.



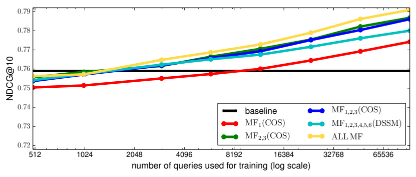
(a) Latent Dirichlet Allocation (LDA)



(b) Word-based Translation Model (WTM)



(c) Discriminative Projection Model (DPM)



(d) Deep Structured Semantic Model (DSSM)

Figure 7.7: Performance of the latent semantic models with different metafeatures for different sizes of training data. (Best viewed in color.)

A first general observation from Table 7.3 is that for every latent semantic model, the

Table 7.3: Performance of the ranking models based on the latent semantic models with different metafeatures (10-fold cross-validation). Highest scores per latent semantic model are indicated in boldface. The improvements of models with metafeatures over their respective baseline models (i.e., LDA, WTM, DPM and DSSM) are statistically significant ($p < 0.001$).

	NDCG			
Global ranker features	@1	@3	@5	@10
LDA	0.552	0.619	0.667	0.741
+ $\mathrm{MF}_{1,2,3}(\mathrm{QL}, 15)$	0.564	0.631	0.680	0.754
+ $\mathrm{MF}_{1,3}(\mathrm{TM})$	0.619	0.679	0.720	0.784
+ ALL MF	**0.625**	**0.684**	**0.725**	**0.787**
WTM	0.560	0.626	0.673	0.746
+ $\mathrm{MF}_{1,2,3}(\mathrm{QL}, 15)$	0.576	0.645	0.691	0.762
+ $\mathrm{MF}_{1}(\mathrm{WTM}, 15, 15)$	0.599	0.660	0.704	0.772
+ $\mathrm{MF}_{2}(\mathrm{WTM})$	0.593	0.655	0.699	0.765
+ $\mathrm{MF}_{1}(\mathrm{WTM}, 15, 15)$	0.605	0.667	0.710	0.775
+ $\mathrm{MF}_{1}(\mathrm{QL}, 15)$				
+ $\mathrm{MF}_{3}(\mathrm{WTM}, 15)$	0.597	0.658	0.701	0.770
+ ALL MF	**0.617**	**0.675**	**0.717**	**0.781**
DPM	0.537	0.602	0.648	0.724
+ $\mathrm{MF}_{1}(\mathrm{COS})$	0.605	0.664	0.706	0.772
+ $\mathrm{MF}_{2,3}(\mathrm{COS})$	0.614	0.675	0.717	**0.782**
+ $\mathrm{MF}_{1,2,3}(\mathrm{COS})$				
$(= \mathrm{ALL\ MF})$	**0.615**	**0.676**	**0.718**	**0.782**
DSSM	0.568	0.640	0.687	0.759
+ $\mathrm{MF}_{1}(\mathrm{COS})$	0.609	0.668	0.710	0.774
+ $\mathrm{MF}_{2,3}(\mathrm{COS})$	0.627	0.685	0.725	0.787
+ $\mathrm{MF}_{1,2,3}(\mathrm{COS})$	0.624	0.683	0.723	0.786
+ $\mathrm{MF}_{1,2,3,4,5,6}(\mathrm{DSSM}, 80)$	0.617	0.673	0.715	0.780
+ ALL MF	**0.634**	**0.691**	**0.731**	**0.791**

addition of all metafeatures defined for that model yields the highest performance. We also see that combinations of metafeatures nearly always lead to performance increases. Let us now turn to each research question individually.

RQ 6.1 (a). We find that all metafeatures defined in §7.3.2 yield statistically significant improvements in NDCG scores at truncation levels 1, 3, 5 and 10 over the underlying rankings produced by the latent semantic models' scores, for all latent semantic models. The relative improvements in NDCG scores at truncation levels 1, 3, 5 and 10 are above 10.2%, 7.8%, 6.4% and 4.2%, respectively (Table 7.4). Hence, our metafeatures have a clear early precision enhancing effect.

RQ 6.1 (b). We find that the best results for each latent semantic model are obtained by the combination of all latent semantic model metafeatures ("ALL MF"). However,

Table 7.4: Relative improvements of the rankings given by the global ranker trained with the individual the latent semantic models' scores and all metafeatures defined for the model over the rankings given by the individual latent sematic models' scores (10-fold cross-validation).

	NDCG			
Latent semantic model	@1	@3	@5	@10
LDA	+13.2%	+10.5%	+8.7%	+6.2%
WTM	+10.2%	+7.8%	+6.5%	+4.7%
DPM	+14.5%	+12.3%	+10.8%	+8.0%
DSSM	+11.6%	+8.0%	+6.4%	+4.2%

combinations of metafeatures do not always lead to performance improvements. In particular, for DPM, the addition of $\mathrm{MF}_1(\mathrm{COS})$ does not lead to a statistically significant improvement over the metafeatures $\mathrm{MF}_{2,3}(\mathrm{COS})$. And for DSSM $\mathrm{MF}_{1,2,3}(\mathrm{COS})$ does not outperform $\mathrm{MF}_{2,3}(\mathrm{COS})$. This can be explained by the fact that the components of $\mathrm{MF}_1(\mathrm{COS})$ are the products of the components of $\mathrm{MF}_2(\mathrm{COS})$ and $\mathrm{MF}_3(\mathrm{COS})$: $\mathrm{MF}_1(\mathrm{COS})$ does not bring any new information to the global ranker trained with $\mathrm{MF}_{2,3}(\mathrm{COS})$, but increases the number of features used by the global ranker, which results in a stronger overfitting of the global ranker.

RQ 6.1 (c). We find that the gains obtained from metafeatures in terms of NDCG@10 are proportional to the logarithm of the size of the training data set: the average Pearson correlation coefficient, $r = 0.860$ (with all values in the interval $[0.825; 0.921]$). For some models, viz. LDA, WTM, DSSM, the performance achieved with a relatively small number of queries in the training set (i.e., < 12500) is below the performance of the baseline (not using metafeatures). This is apparently a result of overfitting, as we observed similar results for the global ranker trained with only one feature—the latent semantic model's score. With more queries added for training, all metafeatures end up beating the baseline, and significantly so.

Together, the above results lead to the conclusion that the proposed metafeatures improve the performance of the latent semantic models on the web document ranking task that we consider. Interestingly, through the addition of metafeatures the ranking of latent semantic models has changed: from DSSM $>$ WTM $>$ LDA $>$ DPM (on all metrics) we went to DSSM $>$ LDA $>$ DPM $\approx$ WTM (again, on all metrics) (Table 7.3). This reveals an important experimental issue: a fair comparison of latent semantic models w.r.t. a document ranking task assumes utilizing all potentially useful knowledge mined during the building of these models.

7.5.2 Experiment 2

The outcomes of Experiment 2 are listed in Table 7.5. We find that the metafeatures yield a statistically significant improvement over the output of the global ranker trained with the latent semantic models' scores only. The relative improvements in NDCG scores at truncation levels 1, 3, 5 and 10 are 7.6%, 6.17%, 5.38% and 3.75%, respectively.

Table 7.5: Performance of the model combinations with the latent semantic models' scores only and the latent semantic models' scores and metafeatures (10-fold cross validation). The improvements are statistically significant ($p < 0.001$).

	NDCG			
Global ranker features	@1	@3	@5	@10
models' outputs	0.604	0.664	0.706	0.772
+ models' metafeatures	**0.650**	**0.705**	**0.744**	**0.801**

Next, we take a closer look at the changes brought about by the inclusion of metafeatures in the combination of latent semantic models. Table 7.6 provides a matrix of the changes in relevance labels for the documents returned in the top position for each query in the evaluation data set by the combination of the latent semantic models' scores only (w/o MF) and by the combination extended with the metafeatures (w/ MF). We counted the number of times that the retrieved document is in each of the five relevance categories, from bad to perfect, for both combinations.

We observe that for 70% of the queries, the relevance labels of the documents returned in the top position did not change. For the remaining queries, the number of cases in which the highest ranked document returned by the combination w/ MF is more relevant than one returned by the combination w/o MF exceeds the number of cases in which the combination w/o MF returns a more relevant document than the combination w/ MF for all relevance label pairs. Furthermore, the relative number of cases in which the combination w/ MF returns a more relevant document than the combination w/o MF is smaller for low-performing queries (for which the retrieved documents' labels fall into *bad*, *fair* or *good* categories) than for high-performing queries (for which the retrieved documents' labels fall are *excellent* or *perfect*).

In sum, metafeatures help to bring high-quality documents to the top position rather than that they help to replace non-relevant documents with partially relevant ones.

7.6 Conclusions and Future Work

In this chapter, we answered the following research question:

RQ 6 How to extract potentially useful information from a trained latent semantic model and how to utilize this information for improving ranking of search results?

We have introduced an approach to increase the effectiveness of latent semantic models on a web document ranking task. The approach is based on so-called *metafeatures*—feature vectors that provide a fine-grained description of a latent semantic model's prediction or some aspects of the prediction. We demonstrated our approach using four latent semantic models, two of which are traditional latent semantic models (Latent Dirichlet Allocation [13, 162] and Word-based Translation Model [10]) used in many IR applications, while the other two are recently proposed latent semantic models (Discriminative Projection Model [52, 173] and Deep Structured Semantic Model [75]) that specialize on the web document ranking task.

Table 7.6: Change matrix of the relevance labels of the highest ranked documents returned by the combination of the latent semantic models' scores only (w/o MF) and the combination of latent semantic models' scores and metafeatures (w/ MF). The cell in row X and column Y contains the number of queries for which the documents ranked on the top positions by the combinations w/o MF and w/ MF belong to categories X and Y, respectively.

		w/ MF				
		bad	fair	good	excellent	perfect
w/o MF	bad	12694	3139	4111	486	534
	fair	3096	14446	6294	1152	931
	good	2943	4816	38396	2656	830
	excellent	206	301	1141	4607	215
	perfect	118	196	239	63	7593

Our experimental results show that through the use of metafeatures the performance of a combination of latent semantic models on the document ranking task can be improved by 7.6% and 3.8% in NDCG scores at truncation levels 1 and 10. Moreover, through the use of metafeatures we manage to achieve better performance for each individual latent semantic model by itself than the traditional way of combining all four latent semantic models' scores.

We believe that a latent semantic model's metafeatures along with its score provide a richer representation of the model's prediction than the model's score by itself. We expect metafeatures to be useful in other applications that use the scores of latent semantic models, e.g., online advertising [21], question answering [11], paraphrase detection [40], and textual entailment [151].

As to the limitations of our study, the large size of some metafeatures might increase the chances of overfitting for many machine learning algorithms. To mitigate the issue, dimensionality reduction techniques, such as Principal Component Analysis (PCA) [82], Autoencoder Neural Networks (AENNs) [38], Restricted Boltzmann Machines (RBM) [65] and t-SNE [152], can be considered.

Our approach can be extended to other models whose scores are used as features for a machine learning algorithm. For instance, in web search, a similar approach can be applied to the PageRank model [127]. The score assigned to a document D by the PageRank model is a sum of features over documents that contain a reference to D. Providing information about components of the sum (e.g., mean, variance) might help the global ranker when it attempts to rank documents that have received very similar PageRank scores.

8 Conclusions

In the beginning of this thesis, we motivated our research by a speculation that understanding people's continuously changing needs is an integral part of intelligence. Specifically, we focused on studying user behavior in web search.

In Chapters 2 and 3, we brought up two new angles to the analysis of click models. First, we showed that click models deteriorate over time if retraining is avoided. Second, we showed that click models trained with suboptimal hyperparameters are prone to predict badly calibrated click probabilities. To combat these problems, we adapted Online EM techniques and isotonic regression. And based on our experimental results, we suggested changes to the click model evaluation protocol. We anticipate that the aforementioned effects would be important for other user behavior models. And we hope that our approaches would serve as a good starting point for dealing with similar challenges in the future.

In Chapter 4, we pioneered the idea of modeling the user's search journey as a sequence of distributed vector representations, which we also exploited in Chapters 5 and 6. We proposed three neural network-based models, NCM, CSM and CATM, that use distributed vector representations to explain and predict (i) clicks on a SERP, (ii) sequences of clicks on a SERP, and (iii) times between user actions, such as time-to-first-click, time-to-last-click, time-between-clicks and time-to-abandoned-query. We believe that the ideas underlying these models can be reused for modeling (i) other information interaction events, (ii) sequences of other information interaction events, and (iii) times between other information interaction events. This altogether lays the foundation for constructing a more realistic simulator of user search behavior than those available today, which in turn would open the door to innovations in search algorithms and provide new opportunities to improve our understanding of human information interaction behavior.

In Chapter 7, we presented guidelines for extracting potentially useful information about the latent semantic model's prediction. And we showed that using this information helps to increase the effectiveness of several latent semantic models in web search. We believe that the proposed general approach can be applied to other machine learning (ML) applications.

Below, we provide answers to the research questions posed in §1.1 and suggest possible directions for future work.

8.1 Main Findings

In Chapter 2, we studied click models in an online scenario, where a search engine deals with a stream of click/skip observations. We focused on the most widely used class of click models, i.e., probabilistic graphical model (PGM)-based click models, and the most effective technique for inferring their parameters, i.e., expectation-maximization (EM). We showed empirically that click models deteriorate over time if retraining is avoided and asked the following research question:

RQ 1 How to keep click models up to date with changes in search engine algorithms and user preferences?

We recognized two challenges in keeping click models up to date. First, how to efficiently incorporate newly observed information into a trained click model? Second, how to remove outdated information from a trained click model? We proposed Online EM to efficiently update click models on the fly using readily available EM equations as well as the EM with Forgetting method to deal with outdated click information by discounting past observations depending on their age. Our experiments show that Online EM is orders of magnitude more efficient than retraining the model from scratch using standard EM, while losing little in quality. And that EM with Forgetting surpasses the performance of complete retraining while being as efficient as Online EM.

In Chapter 3, we examined the calibration properties of PGM-based click models. Specifically, we asked the following research question:

RQ 2 Does calibration help to improve click model performance and make it less dependent on the choice of hyperparameters?

We proposed to use isotonic regression to ensure that click probabilities predicted by a click model match the proportion of clicks (at each position) in the held-out data. Our experiments show that (i) isotonic regression significantly improves click models trained with suboptimal hyperparameters in terms of perplexity; and that (ii) calibrated click models are less sensitive to the choice of hyperparameters than their original (non-calibrated) versions. Importantly, the relative ranking of existing click models in terms of their predictive performance changes depending on whether or not we calibrate their predictions. We advocate for making calibration a mandatory part of the click model evaluation protocol.

In Chapter 4, we introduced an alternative to PGM-based click models. We described a neural click modeling framework, in which user behavior is modeled as a sequence of distributed vector representations, and asked the following research question:

RQ 3 How to design a neural network that would be able to learn patterns in user click behavior directly from logged interaction data?

We instantiated the proposed neural click modeling framework using recurrent neural networks (RNNs). Our experimental results show that the neural click model that uses the same training data as traditional PGM-based click models, has better performance on the click prediction task (i.e., predicting user clicks on search engine results) and the relevance prediction task (i.e., ranking documents by their relevance to a query). We also showed how to go beyond the level of query-document pairs and incorporate information (i) about all query sessions generated by a given query and (ii) about all query sessions containing a given document. The first yields a small improvement on the click prediction task and a considerable improvement on the relevance prediction task. The second yields a large improvement on the click prediction task and a significant deterioration on the relevance prediction task. Finally, we provided an analysis of the best performing model on the click prediction task, which showed that (i) it learned concepts that are used in existing PGM-based click models; and that (ii) it also learned other concepts that cannot be designed manually.

In Chapter 5, we investigated the problem of predicting click sequences. Specifically, we asked the following research question:

RQ 4 What are the challenges in predicting sequences of clicks and how to solve them?

Since the number and order of clicks may vary even for the same query and ranking of results (e.g., due to different users and contexts), there exists no unique *correct* click sequence, but a (possibly infinite) set of *probably correct* sequences does exist. Therefore, we formulated our task as building a model that, given a query and a ranking of results, describes these probably correct click sequences. We proposed a click sequence model (CSM) that predicts a probability distribution over click sequences. Our experiments show that the K most probable click sequences predicted by CSM provide a good means to reason about properties of the probably correct click sequences, such as the expected number of clicks, the expected order of clicks and the probability that a user will interact with the results in a non-sequential order. Moreover, we observed that CSM reaches state-of-the-art performance on the task of predicting clicks, outperforming PGM-based click models by a large margin, and matching the results of NCM described in Chapter 4.

In Chapter 6, we addressed the problem of modeling times between user actions. Specifically, we asked the following research question:

RQ 5 How to correctly interpret times between user actions observed in different contexts?

We introduced the notion of *context bias effect* in times between actions and proposed a context-aware time model (CATM) that allows us to estimate parameters of a probability distribution of the time elapsed between user actions in a given context. We found that for between 37% to 80% of query-result pairs, depending on the number of observations, the distributions of click dwell times have statistically significant differences in query sessions for which a given result (i) is the first item to be clicked and (ii) is not the first. Similarly, we showed that previous user interactions influence distributions of times between (i) submission of a query and the first click on a SERP, (ii) submission of a query and the last click on a SERP, and (iii) submission of an abandoned query (i.e., a query with no clicks on a SERP) and the next query submission. Our experiments show that CATM has better prediction performance than the standard approach that fits a probability distribution to the observed times. And that CATM provides a better means to infer result relevance from times between clicks than the standard approach that computes the mean values of the observed times between clicks.

In Chapter 7, we shifted our attention from interaction-based models towards content-based models. We aimed to improve the effectiveness of the best performing latent semantic models, which utilize user behavioral signals for training. But we also wanted to test the generalizability of our methods, and for this reason we also applied it to unsupervised latent semantic models. Specifically, we asked the following research question:

RQ 6 How to extract potentially useful information from a trained latent semantic model and how to utilize this information for improving ranking of search results?

We proposed to create *metafeatures*—feature vectors that describe the structure of the model's prediction for a given query-document pair—and pass them to the global ranker along with the models' scores. We provided simple guidelines to represent the latent semantic model's prediction with more than a single value, and illustrated these guidelines using several latent semantic models. Our experimental results show that the proposed metafeatures help to improve the effectiveness of (i) the individual latent semantic models and (ii) a combination of these latent semantic models.

8.2 Future Work

We look back at the research chapters and outline possible directions for future work.

8.2.1 PGM-based models

In Chapters 2 and 3, we asked two research questions: (i) how to keep PGM-based click models up to date and (ii) how to ensure that the predictions of PGM-based click models are well calibrated. We believe that these questions are relevant not only for the PGM-based click models we studied but also for all other models of user behavior (those that have already been proposed and the ones that will be proposed in the future). Therefore, we encourage authors of future studies to consider these questions when a new user model is proposed.

The calibration methodology proposed in Chapter 3 can be applied both to user models that predict binary information interaction events, such as clicks, and to user models that predict real value characteristics of information interaction events, such as click dwell times. We do not follow this methodology in Chapters 2, 4, 5 and 6, because the research described in these chapters was conducted before we advocated that models of user behavior need to be calibrated to ensure a fair comparison. We expect the neural network-based user models proposed in Chapters 4, 5 and 6 to suffer less from the issue of bad calibration than the PGM-based click models studied in Chapter 3, because neural networks tend to produce well-calibrated probabilities [123]. We leave the analysis of their calibration properties and a fair comparison with the calibrated PGM-based click models for future work.

8.2.2 Neural network-based models

In Chapters 4, 5 and 6, we studied neural methods for modeling (i) clicks on a SERP, (ii) sequences of clicks on a SERP, and (iii) times between user actions (e.g., time between two consecutive clicks). We formulate the next research goal as building a *universal user model* with the following properties:

- capable of learning from raw data of any form;
- capable of predicting all information interaction events and their properties;
- highly personalized and contextualized; and
- capable of explaining its predictions and providing insights on how to improve the user experience.

Learning from raw data of any form

The models presented in Chapters 4 and 5, NCM and CSM, represent queries and search engine results by their IDs and make predictions using only click/skip information associated with the ID of the query submitted by a user and the IDs of the search engine results. This leads to high-quality predictions for SERPs consisting of frequent query-result pairs and low-quality predictions for SERPs with rare or previously unseen query-result pairs. The CATM presented in Chapter 6 also takes into account the number of terms in the query and the BM25 [136] scores computed between consecutive queries in a search session.

Future user models should be capable of using all potentially useful information, which includes, but is not limited to, (i) the user's query, (ii) a description of each result (URL, title, snippet, main content, images on the landing page, etc.), (iii) the SERP layout, (iv) detailed user interactions with the results (clicks, touches, swipes, mouse tracks, etc.), (v) a user profile, (vi) the user context, and (vii) previous user interactions. Moreover, this information should be presented in raw form (as a list of texts, images and audio), because neural networks have been shown to achieve their best performance when they are applied to raw data.

Predicting all information interaction events and their properties

In Chapters 4, 5 and 6, we formulated the task as predicting (i) clicks on a SERP, (ii) sequences of clicks on a SERP, and (iii) times between clicks and query submissions.

Future user models should be capable of predicting (i) all information interaction events, such as touches (on mobile), swipes (on mobile), mouse movements (on desktop), next queries, etc., (ii) sequences consisting of all information interaction events, and (iii) times between all information interaction events.

Personalization and contextualization

The models proposed in Chapters 4, 5 and 6 do not use any information about the user.

Future user models should be capable of exploiting all potentially useful information about the user, which includes, but is not limited to, (i) demographic information (age, gender, occupation, etc.), (ii) social network profile, circle and recent posts, (iii) short- and long-term user interests and preferences, and (iv) location, time of the day, news and nearby places.

Providing explanations and insights

In Chapters 4, 5 and 6, we put a strong focus on predicting future user behavior. An important direction for future work is to provide interpretable explanations for user models' predictions. For example, a user is likely to be satisfied by a search result (i) because he likes 90% of the articles published on this web site; (ii) because this article describes a nearby event; or (iii) because this article is similar to a few web pages he spent a long time reading last week. Future studies of user models should also go beyond individual explanations and help us gain insights into what leads to a suboptimal user experience.

8.2.3 Metafeatures

In Chapter 7, we presented the idea of complementing predictions of latent semantic models with so-called metafeatures that describe the structure of latent semantic models' predictions. We believe that this general approach can be applied to other models whose scores are used as features for ML algorithms. For instance, a score assigned to a document D by the PageRank model [127] is used, together with the predictions of latent semantic models, as a feature for the global ranker. It is computed as a sum over documents that contain a reference to D. Providing information about components of the sum (e.g., mean, variance) might help the global ranker when it attempts to rank documents that have received very similar PageRank scores.

Bibliography

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pages 19–26. ACM, 2006. (Cited on pages 1, 3, 70, 76, 78, 93, 94, and 95.)

[2] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR*, pages 3–10. ACM, 2006. (Cited on pages 78, 93, 94, and 95.)

[3] J. Arguello. Predicting search task difficulty. In *Advances in Information Retrieval*, pages 88–99. Springer, 2014. (Cited on pages 3, 76, 77, 78, 93, 94, 95, and 96.)

[4] M. Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, pages 641–647, 1955. (Cited on page 23.)

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. (Cited on page 59.)

[6] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Supervised semantic indexing. In *CIKM*, pages 187–196. ACM, 2009. (Cited on pages 4, 98, 100, 101, and 111.)

[7] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Learning to rank with (a lot of) word features. *Information Retrieval*, 13(3):291–314, 2010. (Cited on page 101.)

[8] Y. Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1): 1–127, 2009. (Cited on pages 32, 36, 38, 84, and 109.)

[9] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. (Cited on pages 37, 38, 39, 59, 60, and 84.)

[10] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *SIGIR*, pages 222–229. ACM, 1999. (Cited on pages 98, 100, 101, 104, 111, 112, and 117.)

[11] A. Berger, R. Caruana, D. Cohn, D. Freitag, and V. Mittal. Bridging the lexical chasm: Statistical approaches to answer-finding. In *SIGIR*, pages 192–199. ACM, 2000. (Cited on page 118.)

[12] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A CPU and GPU math expression compiler. In *SciPy*, 2010. (Cited on page 39.)

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003. (Cited on pages 4, 99, 104, 112, and 117.)

[14] A. Borisov and I. Galinskaya. Yandex school of data analysis Russian-English machine translation system for WMT14. In *WMT*, pages 66–70, 2014.

[15] A. Borisov, J. Dlougach, and I. Galinskaya. Yandex school of data analysis machine translation systems for WMT13. In *WMT*, pages 99–103, 2013.

[16] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A context-aware time model for web search. In *SIGIR*, pages 205–214. ACM, 2016. (Cited on pages 53, 70, 71, and 75.)

[17] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A neural click model for web search. In *WWW*, pages 531–541. International World Wide Web Conferences Steering Committee, 2016. (Cited on pages 29, 71, and 94.)

[18] A. Borisov, P. Serdyukov, and M. de Rijke. Using metafeatures to increase the effectiveness of latent semantic models in web search. In *WWW*, pages 1081–1091. International World Wide Web Conferences Steering Committee, 2016. (Cited on page 97.)

[19] A. Borisov, J. Kiseleva, I. Markov, and M. de Rijke. Calibration: A simple way to improve click models. In *CIKM*. ACM, 2018. (Cited on page 19.)

[20] A. Borisov, M. Wardenaar, I. Markov, and M. de Rijke. A click sequence model for web search. In *SIGIR*, pages 45–54. ACM, 2018. (Cited on page 53.)

[21] A. Broder, M. Fontoura, V. Josifovski, and L. Riedel. A semantic approach to contextual advertising. In *SIGIR*, pages 559–566. ACM, 2007. (Cited on page 118.)

[22] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah. Signature verification using a "Siamese" time delay neural network. *Int. J. Pattern Recognition and Artificial Intelligence*, 7(4):669–688, 1993. (Cited on pages 98, 100, and 101.)

[23] M. Burtsev, A. Chuklin, J. Kiseleva, and A. Borisov. Search-oriented conversational AI (SCAI). In *ICTIR*, pages 333–334. ACM, 2017.

[24] G. Buscher, R. W. White, S. Dumais, and J. Huang. Large-scale analysis of individual and task differences in search result page examination strategies. In *WSDM*, pages 373–382. ACM, 2012.

(Cited on pages 78, 94, and 96.)

[25] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, pages 875–883. ACM, 2008. (Cited on pages 1 and 70.)

[26] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Yahoo! Learning to Rank Challenge*, pages 1–24, 2011. (Cited on pages 3, 76, 78, 93, 94, and 95.)

[27] O. Chapelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *WWW*, pages 1–10. ACM, 2009. (Cited on pages 1, 9, 14, 22, 25, 30, 31, 41, 62, 71, 72, and 93.)

[28] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *CIKM*, pages 621–630. ACM, 2009. (Cited on page 30.)

[29] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *TOIS*, 30(1):6, 2012. (Cited on pages 3, 70, 76, 93, 94, and 95.)

[30] D. Chen, W. Chen, H. Wang, Z. Chen, and Q. Yang. Beyond ten blue links: Enabling user click modeling in federated web search. In *WSDM*, pages 463–472. ACM, 2012. (Cited on pages 2, 32, and 94.)

[31] W. Chen, D. Wang, Y. Zhang, Z. Chen, A. Singla, and Q. Yang. A noise-aware click model for web search. In *WSDM*, pages 313–322. ACM, 2012. (Cited on pages 31 and 32.)

[32] A. Chuklin, P. Serdyukov, and M. de Rijke. Click model-based information retrieval metrics. In *SIGIR*, pages 493–502. ACM, 2013. (Cited on pages 9 and 30.)

[33] A. Chuklin, P. Serdyukov, and M. de Rijke. Using intent information to model user behavior in diversified search. In *Advances in Information Retrieval*, pages 1–13. Springer, 2013. (Cited on pages 32 and 94.)

[34] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool, 2015. (Cited on pages 2, 3, 9, 10, 13, 14, 19, 20, 22, 25, 30, 41, 51, 53, 56, 71, 75, 83, and 94.)

[35] W. J. Conover. *Practical Nonparametric Statistics*. Wiley, 1980. (Cited on page 87.)

[36] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94. ACM, 2008. (Cited on pages 1, 2, 22, 31, 70, 71, and 94.)

[37] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990. (Cited on pages 4, 98, and 99.)

[38] D. DeMers and G. Cottrell. Non-linear dimensionality reduction. In *NIPS*, pages 580–580, 1993. (Cited on page 118.)

[39] F. Diaz, R. White, G. Buscher, and D. Liebling. Robust models of mouse movement on dynamic web search results pages. In *CIKM*, pages 1451–1460. ACM, 2013. (Cited on pages 71 and 94.)

[40] B. Dolan, C. Quirk, and C. Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING 2004*, pages 350–356, 2004. (Cited on page 118.)

[41] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55 (10):78–87, 2012. (Cited on page 99.)

[42] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *ICML*, pages 105–112. Morgan Kaufmann, 1996. (Cited on page 21.)

[43] S. T. Dumais. The web changes everything: Understanding and supporting people in dynamic information environments. In *ECDL*, page 1. Springer, 2010. (Cited on page 2.)

[44] G. Dupret and M. Lalmas. Absence time and user engagement: Evaluating ranking functions. In *WSDM*, pages 173–182. ACM, 2013. (Cited on page 94.)

[45] G. Dupret and C. Liao. A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In *WSDM*, pages 181–190. ACM, 2010. (Cited on pages 9 and 30.)

[46] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR*, pages 331–338. ACM, 2008. (Cited on pages 1, 14, 25, 26, 30, 31, 40, 62, 71, and 72.)

[47] J. L. Elman. *Rethinking innateness: A connectionist perspective on development*, volume 10. MIT press, 1998. (Cited on page 32.)

[48] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *TOIS*, 23(2):147–168, 2005. (Cited on pages 3, 70, 76, 79, and 92.)

[49] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001. (Cited on page 112.)

[50] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987. (Cited on page 98.)

[51] J. Gao, X. He, and J.-Y. Nie. Clickthrough-based translation models for web search: From word models to phrase models. In *SIGIR*, pages 1139–1148. ACM, 2010. (Cited on pages 98, 100, 101, 104,

and 111.)

[52] J. Gao, K. Toutanova, and W.-t. Yih. Clickthrough-based latent semantic models for web search. In *SIGIR*, pages 675–684. ACM, 2011. (Cited on pages 4, 98, 100, 101, 104, 111, 112, and 117.)

[53] T. Graepel, J. Quiñonero Candela, T. Borchert, and R. Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *ICML*, pages 13–20. Omnipress, 2010. (Cited on page 1.)

[54] A. Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012. (Cited on page 60.)

[55] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *IEEE Transactions on Neural Networks*, 18(5):602–610, 2005. (Cited on page 38.)

[56] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649. IEEE, 2013. (Cited on pages 32 and 84.)

[57] A. Grotov, A. Chuklin, I. Markov, L. Stout, F. Xumara, and M. de Rijke. A comparative study of click models for web search. In *CLEF*, pages 78–90. Springer, 2015. (Cited on pages 10, 14, 19, 22, 41, and 62.)

[58] Z. Guan and E. Cutrell. An eye tracking study of the effect of target rank on web search. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 417–420. ACM, 2007. (Cited on pages 2 and 22.)

[59] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW*, pages 11–20. ACM, 2009. (Cited on pages 1, 14, 31, 62, and 71.)

[60] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM*, pages 124–131. ACM, 2009. (Cited on pages 1, 25, 31, 62, 70, 71, and 72.)

[61] A. Hassan. A semi-supervised approach to modeling web search satisfaction. In *SIGIR*, pages 275–284. ACM, 2012. (Cited on pages 3, 70, 76, 78, 93, 94, 95, and 96.)

[62] A. Hassan, R. Jones, and K. L. Klinkner. Beyond DCG: User behavior as a predictor of a successful search. In *WSDM*, pages 221–230. ACM, 2010. (Cited on pages 78 and 96.)

[63] A. Hassan, R. W. White, S. T. Dumais, and Y.-M. Wang. Struggling or exploring?: Disambiguating long search sessions. In *WSDM*, pages 53–62. ACM, 2014. (Cited on pages 3, 56, 71, 76, 78, 94, 95, and 96.)

[64] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, and S. Bowers. Practical lessons from predicting clicks on ads at facebook. In *ADKDD*, pages 1–9. ACM, 2014. (Cited on page 1.)

[65] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. (Cited on page 118.)

[66] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on pages 37, 38, and 83.)

[67] K. Hofmann, S. Whiteson, and M. de Rijke. Estimating interleaved comparison outcomes from historical click data. In *CIKM*, pages 1779–1783. ACM, October 2012. (Cited on page 70.)

[68] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval Journal*, 16(1):63–90, 2013. (Cited on page 55.)

[69] K. Hofmann, L. Li, and F. Radlinski. Online evaluation for information retrieval. *Foundations and Trends® in Information Retrieval*, 10(1):1–117, 2016. (Cited on pages 1 and 2.)

[70] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57. ACM, 1999. (Cited on pages 4, 99, and 104.)

[71] V. Hollink, J. Kamps, C. Monz, and M. de Rijke. Monolingual document retrieval for European languages. *Information Retrieval*, 7:33–52, 2004. (Cited on page 109.)

[72] G. Howard. Frames of mind: The theory of multiple intelligences. *NY: Basics*, 1983. (Cited on page 1.)

[73] J. Huang, R. W. White, and S. Dumais. No clicks, no problem: Using cursor movements to understand and improve search. In *SIGCHI*, pages 1225–1234. ACM, 2011. (Cited on pages 53 and 94.)

[74] J. Huang, R. W. White, G. Buscher, and K. Wang. Improving searcher models using mouse cursor activity. In *SIGIR*, pages 195–204. ACM, 2012. (Cited on page 94.)

[75] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338. ACM, 2013. (Cited on pages 4, 98, 100, 101, 107, 108, 111, 112, and 117.)

[76] B. Huurnink, L. Hollink, W. van den Heuvel, and M. de Rijke. Search behavior of media professionals

at an audiovisual archive: A transaction log analysis. *JASIST*, 61(6):1180–1197, June 2010. (Cited on page 70.)

[77] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48. ACM, 2000. (Cited on pages 41, 87, and 112.)

[78] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011. (Cited on page 72.)

[79] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng. Learning user reformulation behavior for query auto-completion. In *SIGIR*, pages 445–454. ACM, 2014. (Cited on pages 1 and 70.)

[80] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142. ACM, 2002. (Cited on pages 1, 22, 70, 93, and 94.)

[81] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*, pages 154–161. ACM, 2005. (Cited on pages 1, 2, 22, and 94.)

[82] I. Jolliffe. *Principal Component Analysis*. Wiley Online Library, 2005. (Cited on page 118.)

[83] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, pages 2342–2350, 2015. (Cited on page 38.)

[84] T. Kenter, A. Borisov, and M. de Rijke. Siamese CBOW: optimizing word embeddings for sentence representations. In *ACL*, pages 941–951, 2016.

[85] T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *SIGIR*, pages 1403–1406. ACM, 2017.

[86] T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *ECIR*, page 837. Springer, 2018.

[87] T. Kenter, A. Borisov, C. Van Gysel, M. Dehghani, M. de Rijke, and B. Mitra. Neural networks for information retrieval. In *WSDM*, pages 779–780. ACM, 2018.

[88] Y. Kim, A. Hassan, R. W. White, and I. Zitouni. Comparing client and server dwell time estimates for click-level satisfaction prediction. In *SIGIR*, pages 895–898. ACM, 2014. (Cited on page 78.)

[89] Y. Kim, A. Hassan, R. W. White, and I. Zitouni. Modeling dwell time to predict click-level satisfaction. In *WSDM*, pages 193–202. ACM, 2014. (Cited on pages 3, 70, 71, 76, 78, 79, 81, 82, 94, 95, and 96.)

[90] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015. (Cited on page 60.)

[91] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. (Cited on page 28.)

[92] J. Kiseleva and M. de Rijke. Evaluating personal assistants on mobile devices. In *CAIR*. ACM, 2017. (Cited on page 70.)

[93] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: Survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181, 2009. (Cited on pages 1 and 2.)

[94] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *KDD*, pages 1168–1176. ACM, 2013. (Cited on page 1.)

[95] D. Koller and N. Friedman. *Probabilistic graphical models: Principles and techniques*. MIT press, 2009. (Cited on pages 30 and 31.)

[96] E. Kravi, I. Guy, A. Mejer, D. Carmel, Y. Maarek, D. Pelleg, and G. Tsur. One query, many clicks: Analysis of queries with multiple clicks by the same user. In *CIKM*, pages 1423–1432. ACM, 2016. (Cited on page 54.)

[97] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. (Cited on page 32.)

[98] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, pages 111–119. ACM, 2001. (Cited on page 105.)

[99] D. Lefortier, P. Serdyukov, and M. de Rijke. Online exploration for detecting shifts in fresh intent. In *CIKM*, pages 589–598. ACM, 2014. (Cited on page 2.)

[100] H. Li and J. Xu. Semantic matching in search. *Foundations and Trends in Information Retrieval*, 7(5): 343–469, 2014. (Cited on page 98.)

[101] Y. Li, A. Dong, H. Wang, H. Deng, Y. Chang, and C. Zhai. A two-dimensional click model for query auto-completion. In *SIGIR*, pages 455–464. ACM, 2014. (Cited on pages 1 and 70.)

[102] P. Liang and D. Klein. Online EM for unsupervised models. In *NAACL*, pages 611–619, 2009. (Cited on page 12.)

[103] S. Lichtenstein, B. Fischhoff, and L. D. Phillips. Calibration of probabilities: The state of the art. In *Decision Making and Change in Human Affairs*, pages 275–324. Springer, 1977. (Cited on page 21.)

[104] C. Liu, F. Guo, and C. Faloutsos. BBM: Bayesian browsing model from petabyte-scale data. In *KDD*,

pages 537–546. ACM, 2009. (Cited on pages 1 and 9.)

[105] C. Liu, F. Guo, and C. Faloutsos. Bayesian browsing model: Exact inference of document relevance from petabyte-scale data. *ACM Transactions on Knowledge Discovery from Data*, 4(4):Article 19, 2010. (Cited on page 9.)

[106] C. Liu, R. W. White, and S. Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In *SIGIR*, pages 379–386. ACM, 2010. (Cited on pages 3, 71, 76, 79, 81, 82, 87, and 95.)

[107] C. Liu, J. Liu, N. Belkin, M. Cole, and J. Gwizdka. Using dwell time as an implicit measure of usefulness in different task types. *JASIST*, 48(1):1–4, 2011. (Cited on pages 3, 76, 78, 94, 95, and 96.)

[108] J. Liu, C. Liu, M. Cole, N. J. Belkin, and X. Zhang. Exploring and predicting search task difficulty. In *CIKM*, pages 1313–1322. ACM, 2012. (Cited on pages 3, 76, 78, 94, 95, and 96.)

[109] Y. Liu, C. Wang, K. Zhou, J. Nie, M. Zhang, and S. Ma. From skimming to reading: A two-stage examination model for web search. In *CIKM*, pages 849–858. ACM, 2014. (Cited on page 31.)

[110] Y. Liu, X. Xie, C. Wang, J.-Y. Nie, M. Zhang, and S. Ma. Time-aware click model. *TOIS*, 35(3):16, 2017. (Cited on page 71.)

[111] S. Malkevich, I. Markov, E. Michailova, and M. de Rijke. Evaluating and analyzing click simulation in web search. In *ICTIR*, pages 281–284. ACM, 2017. (Cited on page 9.)

[112] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. (Cited on page 107.)

[113] I. Markov, E. Kharitonov, V. Nikulin, P. Serdyukov, M. de Rijke, and F. Crestani. Vertical-aware click model-based effectiveness metrics. In *CIKM*, pages 1867–1870, 2014. (Cited on page 9.)

[114] I. Markov, A. Borisov, and M. de Rijke. Online expectation-maximization for click models. In *CIKM*, pages 2195–2198. ACM, 2017. (Cited on page 9.)

[115] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, and D. Golovin. Ad click prediction: A view from the trenches. In *KDD*, pages 1222–1230. ACM, 2013. (Cited on page 1.)

[116] M. R. Meiss, F. Menczer, S. Fortunato, A. Flammini, and A. Vespignani. Ranking web sites with real user traffic. In *WSDM*, pages 65–76. ACM, 2008. (Cited on page 1.)

[117] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010. (Cited on pages 32 and 84.)

[118] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013. (Cited on page 32.)

[119] N. Mishra, R. W. White, S. Ieong, and E. Horvitz. Time-critical search. In *SIGIR*, pages 747–756. ACM, 2014. (Cited on pages 78 and 96.)

[120] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP*, 2(331-340):2, 2009. (Cited on page 72.)

[121] A. H. Murphy and R. L. Winkler. Reliability of subjective probability forecasts of precipitation and temperature. *Applied Statistics*, 26(1):41–47, 1977. (Cited on pages 20 and 21.)

[122] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. MIT Press, 1999. (Cited on pages 10, 11, and 12.)

[123] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *ICML*, pages 625–632. ACM, 2005. (Cited on pages 20, 21, 23, 24, and 122.)

[124] D. Oard and J. Kim. Implicit feedback for recommender systems. In *Proceedings of the AAAI Workshop on Recommender Systems*. AAAI, 1998. (Cited on page 70.)

[125] D. Odijk, R. W. White, A. H. Awadallah, and S. T. Dumais. Struggling and success in web search. In *CIKM*, pages 1551–1560. ACM, 2015. (Cited on pages 3, 54, 56, 71, 76, 78, 94, 95, and 96.)

[126] N. O'Hare, P. de Juan, R. Schifanella, Y. He, D. Yin, and Y. Chang. Leveraging user interaction signals for web image search. In *SIGIR*, pages 559–568. ACM, 2016. (Cited on pages 1 and 70.)

[127] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. (Cited on pages 1, 118, and 124.)

[128] S. Pandey and C. Olston. User-centric web crawling. In *Proceedings of the 14th international conference on World Wide Web*, pages 401–411. ACM, 2005. (Cited on page 1.)

[129] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013. (Cited on page 59.)

[130] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318, 2013. (Cited on pages 39, 60, and 84.)

[131] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood

methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999. (Cited on pages 21 and 24.)

[132] M. J. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964. (Cited on page 112.)

[133] K. Radinsky, K. M. Svore, S. T. Dumais, M. Shokouhi, J. Teevan, A. Bocharov, and E. Horvitz. Behavioral dynamics on the web: Learning, modeling, and prediction. *Transactons on Information Systems*, 31(3):Article 16, 2013. (Cited on pages 2 and 9.)

[134] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM*, pages 43–52. ACM, 2008. (Cited on pages 77, 78, 86, and 93.)

[135] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*, pages 521–530. ACM, 2007. (Cited on page 1.)

[136] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. (Cited on page 123.)

[137] T. Robertson, F. T. Wright, and R. Dykstra. *Order Restricted Statistical Inference*. Probability and Statistics Series. Wiley, 1988. (Cited on pages 20, 21, and 23.)

[138] D. E. Rumelhart, J. L. McClelland, and P. R. Group. *Parallel distributed processing*, volume 1. IEEE, 1988. (Cited on page 32.)

[139] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009. (Cited on page 100.)

[140] A. T. Scaria, R. M. Philip, R. West, and J. Leskovec. The last click: Why users give up information network navigation. In *WSDM*, pages 213–222. ACM, 2014. (Cited on pages 56 and 71.)

[141] A. Schuth, K. Hofmann, and F. Radlinski. Predicting search satisfaction metrics with interleaved comparisons. In *SIGIR*, pages 463–472. ACM, 2015. (Cited on pages 3, 70, 76, 93, 94, and 95.)

[142] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke. Multileave gradient descent for fast online learning to rank. In *WSDM*, pages 457–466. ACM, 2016. (Cited on page 55.)

[143] D. Sculley, R. G. Malkin, S. Basu, and R. J. Bayardo. Predicting bounce rates in sponsored search advertisements. In *KDD*, pages 1325–1334. ACM, 2009. (Cited on page 1.)

[144] S. Shen, B. Hu, W. Chen, and Q. Yang. Personalized click model through collaborative filtering. In *WSDM*, pages 323–332. ACM, 2012. (Cited on pages 32 and 94.)

[145] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, pages 101–110. ACM, 2014. (Cited on pages 4, 98, 100, 101, 111, and 112.)

[146] P. Slovic. *The Perception of Risk*. Routledge, 2016. (Cited on page 21.)

[147] Y. Song, X. Shi, R. White, and A. H. Awadallah. Context-aware web search abandonment prediction. In *SIGIR*, pages 93–102. ACM, 2014. (Cited on pages 78, 94, and 96.)

[148] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014. (Cited on pages 32 and 84.)

[149] K. M. Svore and C. J. Burges. A machine learning approach for improved BM25 retrieval. In *CIKM*, pages 1811–1814, 2009. (Cited on page 111.)

[150] B. Thiesson, C. Meek, and D. Heckerman. Accelerating EM for large databases. *Machine Learning*, 45:279–299, 2001. (Cited on pages 10, 11, and 12.)

[151] P. D. Turney. Measuring semantic similarity by latent relational analysis. In *IJCAI'05*, pages 1136–1141, 2005. (Cited on page 118.)

[152] L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 9(2579-2605):85, 2008. (Cited on pages 42 and 118.)

[153] A. Voronov, A. Borisov, and D. Vatolin. System for automatic detection of distorted scenes in stereo video. In *VPQM*, 2012.

[154] A. Voronov, D. Vatolin, D. Sumin, V. Napadovsky, and A. Borisov. Towards automatic stereo-video quality assessment and detection of color and sharpness mismatch. In *IC3D*. IEEE, 2012.

[155] A. Voronov, D. Vatolin, D. Sumin, V. Napadovsky, and A. Borisov. Methodology of stereoscopic motion picture quality assessment. In *SPIE*. International Society for Optics and Photonics, 2013.

[156] C. Wang, Y. Liu, M. Zhang, S. Ma, M. Zheng, J. Qian, and K. Zhang. Incorporating vertical results into search click models. In *SIGIR*, pages 503–512. ACM, 2013. (Cited on pages 32 and 94.)

[157] C. Wang, Y. Liu, M. Wang, K. Zhou, J.-y. Nie, and S. Ma. Incorporating non-sequential behavior into click models. In *SIGIR*, pages 283–292. ACM, 2015. (Cited on pages 54, 62, and 71.)

[158] H. Wang, C. Zhai, A. Dong, and Y. Chang. Content-aware click modeling. In *WWW*, pages 1365–1376, 2013. (Cited on pages 31 and 32.)

[159] K. Wang, N. Gloy, and X. Li. Inferring search behaviors using partially observable markov (pom)

model. In *WSDM*, pages 211–220. ACM, 2010. (Cited on page 71.)

[160] Q. Wang, J. Xu, H. Li, and N. Craswell. Regularized latent semantic indexing. In *SIGIR*, pages 685–694. ACM, 2011. (Cited on pages 4, 98, 99, and 101.)

[161] Y. Wang, D. Yin, L. Jie, P. Wang, M. Yamada, Y. Chang, and Q. Mei. Beyond ranking: Optimizing whole-page presentation. In *WSDM*, pages 103–112. ACM, 2016. (Cited on pages 1 and 70.)

[162] X. Wei and W. B. Croft. LDA-based document models for ad-hoc retrieval. In *SIGIR*, pages 178–185. ACM, 2006. (Cited on pages 4, 98, 99, 101, 112, and 117.)

[163] M. West and M. D. Escobar. Hierarchical priors and mixture models, with application in regression and density estimation. Technical report, CMU, 1993. (Cited on page 28.)

[164] R. W. White. *Interactions with Search Systems*. Cambridge University Press, 2016. (Cited on page 70.)

[165] K. Williams and I. Zitouni. Does that mean you're happy?: RNN-based modeling of user interaction sequences to detect good abandonment. In *CIKM*, pages 727–736. ACM, 2017. (Cited on page 71.)

[166] W. Wu, H. Li, and J. Xu. Learning query and document similarities from click-through bipartite graph with metadata. In *WSDM*, pages 687–696. ACM, 2013. (Cited on page 70.)

[167] W. Wu, Z. Lu, and H. Li. Learning bilinear model for matching queries and documents. *JMLR*, 14(1): 2519–2548, 2013. (Cited on pages 4, 98, 100, 101, and 111.)

[168] X. Xie, J. Mao, M. de Rijke, R. Zhang, M. Zhang, and S. Ma. Constructing an interaction behavior model for web image search. In *SIGIR*. ACM, 2018. (Cited on page 71.)

[169] Q. Xing, Y. Liu, J.-Y. Nie, M. Zhang, S. Ma, and K. Zhang. Incorporating user preferences into click models. In *CIKM*, pages 1301–1310, 2013. (Cited on page 9.)

[170] D. Xu, Y. Liu, M. Zhang, S. Ma, and L. Ru. Incorporating revisiting behaviors into click models. In *WSDM*, pages 303–312. ACM, 2012. (Cited on page 70.)

[171] W. Xu, E. Manavoglu, and E. Cantu-Paz. Temporal click model for sponsored search. In *SIGIR*, pages 106–113. ACM, 2010. (Cited on page 71.)

[172] Yandex. Personalized web search challenge. `https://www.kaggle.com/c/yandex-personalized-web-search-challenge`, 2014. (Cited on page 56.)

[173] W.-t. Yih, K. Toutanova, J. C. Platt, and C. Meek. Learning discriminative projections for text similarity measures. In *CoNLL*, pages 247–256. ACL, 2011. (Cited on pages 98, 100, 101, 111, 112, and 117.)

[174] E. Yilmaz, M. Shokouhi, N. Craswell, and S. Robertson. Expected browsing utility for web search evaluation. In *CIKM*, pages 1561–1564. ACM, 2010. (Cited on page 30.)

[175] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *ICML*, pages 609–616. ACM, 2001. (Cited on page 21.)

[176] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *KDD*, pages 694–699. ACM, 2002. (Cited on page 21.)

[177] M. D. Zeiler. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (Cited on pages 39 and 84.)

[178] Y. Zhang, D. Wang, G. Wang, W. Chen, Z. Zhang, B. Hu, and L. Zhang. Learning click models via probit bayesian inference. In *CIKM*, pages 439–448. ACM, 2010. (Cited on page 1.)

[179] Y. Zhang, W. Chen, D. Wang, and Q. Yang. User-click modeling for understanding and predicting search-behavior. In *KDD*, pages 1388–1396. ACM, 2011. (Cited on pages 2, 32, and 94.)

[180] Y. Zhang, H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu. Sequential click prediction for sponsored search with recurrent neural networks. *arXiv preprint arXiv:1404.5772*, 2014. (Cited on page 71.)

[181] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *TOMS*, 23(4):550–560, 1997. (Cited on page 85.)

Summary

Web search engines provide quick and easy access to information available online. In the early days of the Internet, web links were the most valuable source of information for predicting result usefulness. Nowadays, the whole web search stack relies on user behavioral data, starting from crawling policies to optimizing presentation of the results. However, accurately interpreting user interaction behavior is not straightforward due to various types of bias. For example, users tend to click more on results ranked on top positions (position bias) and visually salient results (attention bias). In this thesis, we study existing tools for modeling and predicting user interactions with a search engine, improve them and develop new ways of gaining insights about user behavior.

In the first two research chapters, we focus on the traditional approach to account for biases in user click data, which requires a probabilistic graphical model (PGM) that explains relationships between click/skip events (observed variables) and examination (unobserved variables). We refer to these models as PGM-based click models and bring two new angles to their analysis. First, we show that PGM-based click models deteriorate over time if retraining is avoided and ask how to keep them up to date with changes in search engine algorithms and user preferences. Second, we show that PGM-based click models trained with suboptimal hyperparameters are prone to predict badly calibrated click probabilities and investigate whether calibration helps to improve their prediction performance.

In the next three research chapters, we propose an alternative to the PGM-based approach. Specifically, we suggest to model user click behavior in web search using recurrent neural networks (RNNs). We show that RNN-based click models achieve better prediction performance compared to PGM-based click models. We explain this by the fact that RNN-based click models learn user behavior patterns directly from the data, while in PGM-based click models these patterns need to be set manually. Next to the biases we observe in user click behavior, we find biases in times between user actions. For example, users tend to spend more time interacting with results at the beginning of a search session. And we show that RNN-based models can account for biases in times between user actions too.

In the last research chapter, we turn our attention towards content-based models or, as they are often referred to, latent semantic models. Today, the use of latent semantic models by search engines is restricted to simply passing their outputs as features to a so-called global ranker, along with outputs of other models used for ranking. We argue that this is not optimal, because a single value output by a latent semantic model may be insufficient to describe all aspects of the latent semantic model's prediction. We present guidelines for extracting potentially useful information about the latent semantic model's prediction. And we show that using this information helps to increase the effectiveness of several latent semantic models in web search.

Samenvatting

Zoekmachines geven snel en gemakkelijk toegang tot informatie op het internet. In de begindagen van het web vormden links tussen documenten de meest waardevolle informatie om te bepalen welke zoekresultaten relevant waren. Tegenwoordig staat de interactie die gebruikers hebben met zoekmachines eerder centraal bij alles wat achter de schermen gebeurt — van het beslissen welke pagina's op te nemen in de index, tot het optimaliseren van de opmaak van de pagina met zoekresultaten. Het interpreteren van gebruikersgedrag is echter niet eenvoudig vanwege verschillende soorten *bias* die zich voordoen. Zo zijn gebruikers geneigd vaker te klikken op resultaten die bovenaan staan in een lijst, en wordt er eerder geklikt op visueel aantrekkelijke resultaten. In dit proefschrift bestuderen we bestaande methodes om de interactie van gebruikers met zoekmachines te voorspellen, verbeteren we deze, en ontwikkelen we nieuwe manieren om inzicht te verkrijgen in het gedrag van gebruikers.

In de eerste twee onderzoekshoofdstukken besteden we aandacht aan traditionele manieren van omgaan met bias in klikdata. We behandelen met name *probabilistic graphical models* (PGMs) die de relatie beschrijven tussen zowel geobserveerde acties, zoals het wel of niet klikken op resultaten, als niet geobserveerde acties, zoals het bekijken van resultaten. We introduceren twee nieuwe inzichten wat betreft de analyse van PGM-gebaseerde klikmodellen. Ten eerste tonen we aan dat klikmodellen slechter worden door de tijd heen als ze niet opnieuw getraind worden, en stellen we ons de vraag hoe de modellen up-to-date gehouden moeten worden wanneer de zoekalgoritmes en het gebruikersgedrag waar ze mee werken veranderen. Ten tweede laten we zien dat een suboptimale keus van hyperparameters bij het trainen van klikmodellen kan leiden tot het voorspellen van slecht gekalibreerde kansen op kliks. We onderzoeken of kalibratie de voorspelde kansen verbetert.

In de drie onderzoekshoofdstukken die volgen stellen we een alternatief voor de PGM-gebaseerde modellen voor, gebaseerd op *recurrent neural networks* (RNNs). We laten zien dat RNN-gebaseerde klikmodellen betere voorspellingen doen dan PGM-gebaseerde modellen. De reden die wij hiervoor aandragen is dat de RNN-gebaseerde modellen de patronen die ze herkennen in gebruikersdata direct leren van de data, terwijl deze bij PGM-gebaseerde modellen handmatig moeten worden gespecificeerd. Bovendien observeren we, naast de *bias* in gebruikershandelingen zelf, een vergelijkbare *bias* in de tijd tussen gebruikershandelingen, bijvoorbeeld de tijd die mensen besteden om naar resultaten te kijken. Ook deze *bias* kan door RNN-gebaseerde modellen worden verklaard.

In het laatste onderzoekshoofdstuk richten we onze aandacht op inhoud-gebaseerde modellen, of, zoals ze vaak worden genoemd, latent semantische modellen. Huidige zoekmachines gebruiken de *output* van latent semantische modellen vaak als *feature*, samen met de *output* van andere modellen, voor een zogenaamde globaal ranking-algoritme. Wij stellen dat dit niet optimaal is, omdat het representeren van de voorspelling van van een latent semantic model als één getal te beperkt zou kunnen zijn om alle aspecten te belichten van de voorspelling van het model. We stellen richtlijnen voor om potentieel bruikbare informatie te onttrekken aan latent semantische modellen. We tonen aan dat het gebruik van deze additionele informatie bijdraagt aan betere resultaten van verscheidene latent semantic modellen voor zoekmachines.

Web search engines provide quick and easy access to information available online. In the early days of the Internet, web links were the most valuable source of information for predicting result usefulness. Nowadays, the whole web search stack relies on user behavioral data, starting from crawling policies to optimizing presentation of the results. However, accurately interpreting user interaction behavior is not straightforward due to various types of bias. For example, users tend to click more on results ranked on top positions (position bias) and visually salient results (attention bias). In this thesis, we study existing tools for modeling and predicting user interactions with a search engine, improve them and develop new ways of gaining insights about user behavior.