Learning Recommender Systems from Biased User Interactions

Jin Huang

Learning Recommender Systems from Biased User Interactions

Jin Huang

Learning Recommender Systems from Biased User Interactions

Jin Huang

Learning Recommender Systems from Biased User Interactions

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 7 februari 2024, te 11.00 uur

door

Jin Huang

geboren te Henan

The research was carried out at the Information Retrieval Lab at the University of Amsterdam.

Copyright © 2024 Jin Huang, Amsterdam, The Netherlands
Cover by Jiawen Zhu
Printed by Ridderprint, The Netherlands

ISBN: 978-94-6483-748-3

Acknowledgements

Over four years ago, I embarked on a journey to Amsterdam to pursue my Ph.D. degree, along, with my backpack, and with a mixture of anticipation and nervousness. Now, nearing the completion of my Ph.D., I feel much more confident and have a clearer sense of who I am and what my future holds. Looking back, this journey was novel, unforgettable, yet extraordinarily challenging, especially as it was significantly affected by Covid-19. I am sincerely grateful to the many people who supported and accompanied me throughout this experience. Without all of you, I would not be where I am today. I want to take this opportunity to express my heartfelt thanks to every one of you.

First and foremost, thank you, Maarten de Rijke, for being an exceptional supervisor. Your invaluable guidance and unwavering support are the cornerstone of my Ph.D. journey. Your mentorship not only shaped the outcome of my thesis but was also instrumental in cultivating my ability to think critically and conduct my research as an independent researcher. Moreover, you supported me in improving my teaching skills and leadership qualities, which had a profound impact on my future career development.

Second, I wish to thank my co-promotors, Herke van Hoof and Harrie Oosterhuis. Herke, you always provided me with valuable insights and constructive suggestions that significantly enhanced the quality of my research work. Harrie, your meticulous attention to detail and commitment to excellence were evident in our weekly meetings, which played a pivotal role in refining my research. I am truly fortunate to have had such a dedicated and supportive team of supervisors, each bringing a unique perspective and expertise to my Ph.D. journey.

It is a great honor to have Hinda Haned, Evangelos Kanoulas, Mounia Lalmas, Sara Magliacane, and Ji-Rong Wen as my Ph.D. committee members. I sincerely appreciate your valuable time devoted to reading and discussing my thesis.

I extend my heartfelt thanks to the individuals who laid the foundation for my academic career and offered crucial assistance during my Ph.D. application process in the past. Thank you, Ji-Rong Wen, Wayne Xin Zhao, and Zhaochun Ren!

I would also like to thank everyone I have met in the IRLab: Ali A, Ali V, Amin, Amir, Ana, Andrew, Anna, Antonis, Arezoo, Arian, Barrie, Chang, Christof, Chuan, Clara, Clemencia, Cosimo, Dan, David, David, Fabio, Gabriel, Gabrielle, Georgios, Hongyi, Hongyu, Ilya, Ivana, Jasmin, Jia-Hong, Jiahuan, Jie, Jingfen, Jingwei, Julien, Kidist, Maarten M, Maartje, Mahsa, Maria, Mariya, Masoud, Maurits, Ming, Mohammad, Mozhdeh, Nikos, Olivier, Pablo, Panagiotis, Peilei, Pengjie, Petra, Philipp, Pooya, Rolf, Romain, Roxana, Ruben, Ruqing, Saedeh, Sam, Sami, Sebastian, Shaojie, Shashank, Simon, Songgaojun, Spyretta, Svitlana, Thilina, Thong, Vaishali, Vera, Wanyu, Weijia, Xiangsheng, Xinyi, Yang, Yangjun, Yibin, Yifan, Yifei, Yongkang, Yuanna, Yuanxing, Yuyue, Zahra, Zihan, Ziming, and Ziyi. Thank you all for making IRLab a great and wonderful group. I really enjoyed our group activities, including but not limited to Soos talks, Research Meetings, SEA talks, reading groups, discussion groups, as well as social events, such as Friday drinks, dumpling parties, outings, and Christmas drinks. Special thanks to Jingfen and Sami for being my paranymphs for my defense. In addition, I would like to thank the students whom I supervised – Bunyamin, Calvin, Cas, Helma, Luke, Margot, and Thijs – for their hard work and for helping me improve my teaching skills.

Many thanks to Sungjin and Ziming for the great opportunity and mentorship you provided during my internship at Amazon in Seattle. Your support has been crucial to my development, and I am grateful for the valuable insights you have shared.

Further thanks to my old friends and new friends. Thank you, Songsong, Xuanxuan, Yangyang, Yueyue, Zizi – my wonderful friends and former roommates from both our bachelor's and master's years. Thank you, Chao, Fei, Gaole, Hongjian, Peng, Tianshi, Weiran, Xiang, Xiaojie, Xiting, Yixing, You, Yulei, Yupeng, and Zhe – my dear friends I met during my bachelor and master studies. Your continuous support during my Ph.D. has meant the world to me. In addition to these lasting friendships, I want to extend my gratitude to the new friends I met during my Ph.D. studies and internship. Thanks to Baohong, Chuxi, Daniel, Eric, Hao, Jiayi, Jie, Kevin, Lili, Na, Ning, Sha, Shiqi, Tao, Teng, Wei, Weiwei, Xiaojuan, Xiaomeng, Yan, Yixian, Yue, Zhudi, and Zhuohao. Your support, laughter, and shared moments have made my journey even more rewarding.

Finally, I want to thank my family. 爸妈，感谢你们一直以来的理解和支持，让我能安心地在荷兰学习和生活，尤其是在艰难的疫情期间。感谢小弟的刻苦和懂事，希望你早日找到自己热爱的事业。感谢姥爷姥姥，总是耐心聆听我讲述自己的研究工作，并时刻关心我的身体健康。祝愿你们健康长寿，平安喜乐！

Jin Huang
Amsterdam
25 December 2023

Contents

1 Introduction

Recommender systems (RSs) are designed to help users quickly narrow down and find what they need from a collection of items [13, 107, 108, 157]. In the past decade, RSs have gained increasing significance across various scenarios, such as e-commerce product recommendations [79, 121], media recommendations [32, 127], news recommendations [84, 167], location-based recommendations [151, 160], and health-and-fitness suggestions [116, 135]. RSs involve predicting user preferences on items and making recommendations to users, primarily based on methods that are learned from user profiles, item features, or user-item interactions [13, 157]. RS methods can be based on a variety of techniques, including collaborative filtering [126], content-based [103], and hybrid approaches [134]. Predominant RS methods rely on supervised learning models to predict user ratings or the probabilities of users interacting with items. These approaches can further be integrated with deep learning models, allowing them to capture complex recommendation patterns [64, 128, 157], such as recurrent neural networks for sequential recommendations [48].

Besides these traditional supervised learning-based RS methods, *reinforcement learning for recommendation* (RL4Rec) is receiving increased attention in both academia and industry [4, 78]. The key idea of reinforcement learning (RL) is to optimize a policy that matches states to actions so that an agent performing these actions achieves maximum cumulative reward [131]. Different from supervised learning, RL does not only consider the immediate reward of an action but also the effect it has on subsequent actions, thus allowing it to learn long-term goals. Leveraging the principles of RL, RL4Rec methods can learn to optimize for long-term user engagement, *e.g.,* the cumulative number of clicks [170]. Typically, RL4Rec methods learn by recommending items to users and observing their subsequent interactions. RL4Rec methods need to explore the item space to avoid falling into sub-optimality [167]. This poses risks when applying RL4Rec online: (1) During learning, exploratory or incorrect actions could be taken and be detrimental to the user experience [72, 75]; and (2) it is time-consuming, costly, and not feasible for many researchers, both in academia and industry, due to their limited access to actual users. As a result, RL4Rec has potential but is not often applied in practice due to these risks.

The predominant RS methods, whether rooted in supervised learning or reinforcement learning, base their learning and evaluation on logged user interactions with items. The ideal but unrealistic user interaction data includes user feedback (*e.g.,* a rating or a click) on all items in a system. In practice, logged user interaction data is often very sparse and subject to heavy selection bias which is a systematic error that arises from data collection [18, 21, 89, 98, 104, 123]. Due to *selection bias*, the process that decides whether a user interacts with an item is not a random selection and certain interactions are much more likely to be observed than others. Two well-known types of selection bias present in user interactions are *popularity bias* [18, 104, 123], where users interact more with popular items, and *positivity bias* [104], where users tend to rate items that they prefer more often. Besides them, there are various forms of bias, including incentive bias [101], conformity bias [70], and two complex biases that we will address in this thesis: *dynamic selection bias* and *multifactorial bias*.

These biases have potentially negative effects on the learning and evaluation of RSs [18, 21, 117]. For instance, if popularity bias is present in user interactions, RS methods learned from such biased logged data might excessively recommend popular items that are over-represented in logged interactions due to popularity bias. Consequently, these methods might disregard individual user preferences and overlook less popular long-tail items. In general, ignoring bias in RSs can lead to various concerns, *e.g.,* over-specialization [3], filter bubbles [95, 102], unfairness [2], and even a decline in user engagement [21]. Therefore, it is important to correct for selection bias when learning RS methods from biased user interactions.

To mitigate the effect of bias in logged user interactions, the task of debiasing RSs has been proposed. Widely-used debiasing methods [62, 117] make use of inverse propensity scoring (IPS) [59] to inversely weight user interactions in the logged data based on their propensities, *i.e.,* the probability of their occurrence due to bias. It assigns higher weights to user interactions that are less likely to occur due to bias and, conversely, assigns lower weights to those that are more likely to occur due to bias. Thereby, this IPS-based debiasing method can correct for the over- or under-representation resulting from selection bias. IPS-based methods base their weights on the propensities of interactions, but these cannot be observed directly and thus require propensity estimation [92, 117]. Bias propensity estimation can be based on the use of naive Bayes with maximum likelihood [18, 117, 158] or on optimizing machine learning models [114, 117]. With the corresponding estimated propensities, the IPS-based debiasing method can mitigate the effect of different forms of bias. Subsequently, an RS method can be debiased when it is optimized by the debiasing method.

Debiasing RSs has emerged with a growing recognition of bias-related issues in RSs and contributed to positive social impacts and more equitable RSs. However, important questions remain.

On the one hand, existing debiasing recommendation methods primarily assume that selection bias remains static or is affected by only one factor, *e.g.,* popularity bias remains unchanged over time and is determined by only the item factor. This stands in contrast to the real world where the popularity of an item may change drastically over time; and user interactions may be subject to multiple combinations of biases or complex biases which are determined by more than one factor [34, 56, 104]. Ignoring these complex forms of bias may lead to misleading predictions of user preferences and recommendations and result in a decline in performance when confronted by actual users.

On the other hand, the effect of bias remains largely unexplored in the domain of RL4Rec. Previous work has proven that bias present in logged user interactions has strong implications for RS methods that learn from these interactions [18, 117]. As a result, we can expect bias to affect RL4Rec methods as they are also learned from these biased interactions. For example, the existing findings on the optimal choice of the state encoder component in RL4Rec methods [83] were derived without taking bias into account and may be altered when bias effects are considered.

In this thesis, we consider different forms of bias in user interactions and investigate the effect of bias on RL4Rec. Specifically, we consider and correct for two forms of bias: *dynamic selection bias* that changes over time and a *multifactorial bias* determined by the item and rating value factors. We will also analyze how the effect of bias in logged data affects RL4Rec simulators and the resulting RL4Rec methods. To mitigate the effect of bias, we will provide the first debiased simulator to enable learning and evaluating RL4Rec methods. Furthermore, we investigate whether the optimal choice of state encoders for RL4Rec methods differs when learning and evaluating using this debiased simulator compared to simulators that ignore bias.

1.1 Research Outline and Questions

This thesis centers on advancing research in the field of debiasing RSs through the following two themes:

(1) Analyzing, estimating, and correcting for two complex forms of selection bias: dynamic selection bias that changes over time and multifactorial bias that is determined by the item and rating value factors; and

(2) Mitigating the effect of bias on RL4Rec methods by introducing a novel debiased simulator and exploring different state encoders for RL4Rec methods when learning and evaluating with this debiased simulator.

1.1.1 Correcting for Complex Forms of Selection Bias

In the first part of the thesis, we consider two complex forms of selection bias, which better reflect real-world scenarios than well-known popularity bias and positivity bias.

As we have pointed out above, selection bias present in user interactions affects the optimization and evaluation of RS methods and thus needs to be corrected [18, 21, 117]. Existing debiasing methods [62, 117] improve recommendations over recommendation approaches that ignore the effect of bias. However, these debiasing methods assume that the effect of selection bias is static over time, despite the fact that selection bias may be dynamic, not static [21, 61]. For instance, movies and news usually experience a surge in attention shortly after being published. Subsequently, their popularity decreases as time goes by [19, 61]. Instead of static selection bias, real-world user behavior may be better captured with dynamic bias.

In Chapter 2, we look at dynamic selection bias present in logged user interactions. Besides selection bias, user preferences may also change over time. We consider a dynamic scenario in which both the selection bias and user preferences are dynamic. Before we introduce our debiasing method for the dynamic scenario, we analyze real-world logged data to verify that the dynamic scenario is realistic:

RQ1 Do we find evidence for dynamic selection bias and dynamic user preferences in real-world data?

We answer this question in the affirmative by looking at the effect of item-age, *i.e.,* the time since the publication of the item, on selection bias and user preferences in the real-world MovieLens dataset [43]. We find that item-age is an essential factor for accurately capturing the selection bias and user preferences in users' behavior in the MovieLens dataset. This confirms that the dynamic scenario does indeed capture real-world data better. We prove that, in the dynamic scenario, the existing static IPS approach is no longer unbiased. Therefore, there is a real need for a method that can deal with the dynamic scenario and Chapter 2 also concerns the question:

RQ2 Can the prevalent IPS-based debiasing method be extended to mitigate the effect of dynamic user selection bias and model dynamic user preferences?

We introduce DANCER, a method for DebiAsing in the dyNamiC scEnaRio. DANCER extends IPS by utilizing propensities that vary per time period, thus enabling the correction of the dynamic effects of selection bias. We further apply DANCER to a time-aware matrix factorization (TMF) method that allows for the modeling of dynamic user preferences, resulting in TMF-DANCER, the first method that is unbiased in the dynamic scenario. Our experimental results indicate that the proposed recommendation approach improves performance in predicting user ratings on items compared to recommendation approaches that build on debiasing methods that incorrectly assume static selection bias in a dynamic scenario.

Besides the dynamic nature of bias, existing debiasing methods also ignore the effect of multiple factors on bias. Instead, they only consider single-factor forms of bias, *e.g.,* only the item (popularity bias) or only the rating value (positivity bias). However, real-world user decisions about interacting with items generally depend on more than one factor [34, 56, 104]. For example, Pradel et al. [104] observed correlations between selection and both popularity and positivity. In addition, as Chapter 2 highlights, selection bias is also affected by the additional factor of time. Hence, a notion of bias that considers the effects of multiple factors may better reflect actual user behavior. Debiasing methods should also be extended to address this richer notion of bias.

In Chapter 3, we consider a multifactorial bias that is determined by two factors: item and rating value. It can be seen as a generalization of both popularity bias and positivity bias that combines the essential properties of both. To mitigate the effect of multifactorial bias, we investigate the following question:

RQ3 Can the IPS-based debiasing method be extended to correct for multifactorial bias?

We use IPS-based optimization with propensity estimation for multifactorial bias and thus derive a debiasing method that corrects for multifactorial bias. Multifactorial bias propensity estimation is crucial for debiasing. We introduce the first propensity estimation method for multifactorial bias that considers both item and rating value factors. It is based on naive Bayes with maximum likelihood and estimates propensities according to logged user ratings and a small sample of user ratings on uniformly randomly selected items. We expect that using the results of our multifactorial bias propensity estimation, a rating prediction model optimized by IPS can correct for multifactorial bias.

While it is intuitive to extend IPS with a corresponding form of propensity estimation to correct for multifactorial bias, this idea comes with severe practical challenges as the consideration of multiple factors greatly increases problems of data sparsity [30, 108]. Even single-factor bias estimation, relying on factor frequencies, already has to deal with severe sparsity. For instance, popularity bias propensity estimates may be inaccurate for less popular items due to limited interactions [30]. Multifactorial bias estimation, which involves the frequencies of combinations of items and rating values, further exacerbates this sparsity problem. As a result, we have to overcome a severe sparsity problem to make our multifactorial method feasible and robust in practice. This leads us to ask the following question in Chapter 3:

RQ4 Can we deal with the severe sparsity problem posed by the multifactorial method?

To answer this question in the affirmative, we propose the adoption of a propensity smoothing technique and a novel alternating gradient descent approach in our multifactorial method. The propensity smoothing technique is adopted in the introduced multifactorial bias propensity estimation method to avoid invalid or extremely small propensity estimates. Simultaneously, the proposed alternating gradient descent approach offers robust and stable optimization for the IPS-debiasing method. Our extensive experimental results show that, upon resolving the sparsity issue, our multifactorial method exhibits significantly enhanced robustness and effectiveness in mitigating the effect of bias compared to previous single-factor debiasing methods.

1.1.2 Learning and Evaluating RL4Rec in a Debiased Simulator

In the second part of the thesis, we are concerned with the effect of selection bias in user interactions on RL4Rec methods that are learned from these interactions.

Despite their potential for improving long-term user engagement, RL4Rec methods are not often applied in practice due to the risks inherent in applying them online. To reduce these risks, we consider simulation-based experiments, which are an alternative to online deployment of RL4Rec [12, 57, 111, 165]. RL4Rec simulators typically simulate user behaviors (*e.g.,* ratings or clicks) on items and allow learning and evaluating RL4Rec methods on these simulated user behaviors. To simulate user behavior while maintaining many of its natural complexities, the simulated user behaviors are usually based on datasets of logged user data and are thus affected by the selection biases present in these user interactions [57, 165]. As a result, RL4Rec methods that are learned with such a simulator would also be affected by bias and may result in detrimental performance if exposed to actual users. Hence, it is crucial to mitigate the effect of bias present in user interactions while constructing a simulator from these biased interactions.

In Chapter 4, we focus on building a simulator for RL4Rec while mitigating the effect of bias. We investigate the following question:

RQ5 Is it possible to mitigate the effect of bias on simulators for RL4Rec?

We propose a debiasing method for RL4Rec simulators that use predicted user-item ratings and a user-choice model on top of the predicted ratings. The proposed debiasing method is intermediate bias mitigation step (IBMS), an intermediate step between the logged data and the learned rating prediction model, and aims to mitigate the bias originating from the data from affecting rating predictions in simulators. By mitigating the effect of bias before the rating prediction model is learned, we are able to minimize its effect to reach subsequent steps, including the final produced RL4Rec methods.

To evaluate how well the proposed IBMS mitigates the effect of bias, a straightforward evaluation approach is to compare the simulated feedback with logged user feedback [26, 119, 165]. Here, we focus on the offline evaluation, as simulators are designed for situations where online deployment is impossible. The downside of this evaluation is that it does not consider the performance of RL4Rec methods learned with the simulator, despite the fact that finding an optimal RL4Rec method is the ultimate goal. Hence, we also ask the following question in Chapter 4:

RQ6 Can the evaluation of a simulator take the performance of the RL4Rec methods that are learned with this simulator into account?

We propose an offline evaluation that does consider the performance of the final produced RL4Rec methods from a simulator. Using our proposed evaluation approach, our experimental results show the effectiveness of the proposed IBMS in mitigating the effect of bias. We further combine the debiasing method IBMS and the newly proposed evaluation method into a novel Simulator for OFfline leArning and evaluation (SOFA), the first that corrects for bias, to help researchers in the field develop and evaluate RL4Rec methods.

The debiased SOFA simulator proposed in Chapter 4 enables the learning and evaluation of RL4Rec methods while mitigating the effect of bias. RL4Rec methods commonly formulate the recommendation task as a Markov decision process (MDP): A state stores a user's historical interactions, an action is to recommend an item to the user, and the reward is the corresponding user feedback. The state encoder is a crucial component of RL4Rec methods and is used to encode a user state into a dense representation, which is used to estimate the user's preference and subsequently guide the RL method in taking actions, *i.e.,* recommending items [83]. We are concerned with reproducing and generalizing existing findings regarding state encoders for RL4Rec methods in our debiased simulated environment. Specifically, we focus on the study of Liu et al. [83], which concluded that an attention-based state encoder leads to the best recommendation performance by comparing with three baseline state encoders in a simulated RL4Rec environment that does not debias logged user data. We investigate the following question in Chapter 5:

RQ7 Can the findings regarding the optimal choice of state encoders in RL4Rec methods generalize to the debiased simulation?

We compare RL4Rec methods with different state encoders in the debiased SOFA simulators introduced in Chapter 4. Besides the debiased simulator, we also generalize the findings to a different RL method, three additional state encoders, and a different dataset. Our experimental results show that Liu et al.'s findings are reproducible in a debiased simulation generated from the same dataset used by Liu et al. [83], *i.e.,* the Yahoo!R3 dataset [89], but they do not generalize to the debiased simulation generated from a different dataset, *i.e.,* the Coat dataset [117]. Moreover, we also find that the attention state encoder, suggested by Liu et al. [83], incurs very high computational costs, but does not always guarantee that the highest performance will be reached.

Below, in Chapters 2–5 we refine some of the thesis-level research questions listed above into one or more chapter-level research questions.

1.2 Main Contributions

In this section, we provide a concise summary of the main algorithmic, theoretical, empirical, and artifact contributions of this thesis as follows:

1.2.1 Algorithmic Contributions

(1) A method for debiasing in the dynamic scenario (DANCER) where both selection bias and user preferences are dynamic.

(2) A propensity estimation method to estimate multifactorial bias; in addition, we introduce a multifactorial debiasing method that extends the existing IPS-based debiasing method by using the multifactorial bias propensity estimates.

(3) An alternating gradient descent approach for robust and stable IPS-debiasing optimization.

(4) An approach for debiasing simulators that mitigates the effect of bias in logged data; in addition, we propose SOFA, a debiased simulator for offline learning and evaluating RL4Rec methods.

(5) An evaluation method to analyze the effect of bias on RL4Rec methods.

(6) Three state encoders in RL4Rec methods to encode user states into dense representations that are used by RL methods to take actions.

1.2.2 Theoretical Contributions

(7) Formal definitions of single-factor bias and multifactorial bias.

(8) A proof that static IPS estimation ignoring dynamic selection bias is not unbiased in dynamic scenarios where both selection bias and user preferences are dynamic.

(9) A formal proof for the unbiasedness of DANCER in dynamic scenarios.

1.2.3 Empirical Contributions

(10) An empirical verification of the existence of dynamic selection bias and dynamic user preferences in real-world data.

(11) An empirical comparison of DANCER with existing debiasing methods designed for static selection bias.

(12) An empirical comparison of our proposed multifactorial debiased method with existing single-factor debiasing methods on real-world datasets and in scenarios where the effect of the item and rating value factors on selection bias is varied.

(13) An empirical demonstration of the effectiveness of our proposed alternating gradient descent optimization method for various debiasing methods on real-world datasets.

(14) An empirical comparison of the SOFA simulator against a naive simulator that ignores the effect of bias present in logged data.

(15) An empirical comparison of RL4Rec methods with different state encoders when learning by interacting with the SOFA simulator.

1.2.4 Artifact Contributions

(16) A debiasing framework for dynamic selection bias correction.

(17) A debiasing framework for multifactorial bias correction.

(18) A debiased simulator for learning and evaluating RL4Rec methods.

(19) An RL4Rec pipeline for standardized implementation while enabling the learning and evaluation of RL4Rec methods using the debiased simulator.

1.3 Thesis Overview

In this section, we provide a brief overview of each chapter of this thesis. This thesis comprises an introduction chapter, which you are currently reading, followed by four research chapters organized into two parts, concluding with a conclusion chapter.

Part I, titled *Correcting for complex forms of selection bias*, consists of two research chapters. Each focuses on one form of selection bias. We start with dynamic selection bias in Chapter 2. This chapter introduces DANCER, a method for debiasing in dynamic scenarios in which both selection bias and user preferences change over time. Chapter 3 looks at multifactorial bias that is determined by two factors: item and rating value, and introduces a multifactorial method to mitigate the effect of multifactorial bias.

Part II, titled *Learning and evaluating RL4Rec in a debiased simulator*, contains two research chapters. Chapter 4 focuses on the effect of bias present in logged data on simulators for offline learning and evaluation and introduces SOFA, a debiased simulator for RL4Rec. Chapter 5 reproduces and generalizes the existing findings regarding state encoders for RL4Rec methods in the debiased SOFA simulator.

Lastly, we wrap up the thesis in Chapter 6. It contains a summary of the findings in this thesis and concludes with a discussion of limitations and prospective future research directions.

1.4 Origins

The research chapters in this thesis are built upon the following publications.

Chapter 2 is based on the following paper:

- J. Huang, H. Oosterhuis, and M. de Rijke. It Is Different When Items are Older: Debiasing Recommendations When Selection Bias and User Preferences are Dynamic. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 381–389. ACM, February 2022.

JH formulated the main research idea, conducted the experiments, and did most of the writing; HO and MdR led the discussions, offered valuable suggestions, and contributed significantly to the text.

Chapter 3 is based on the following paper:

- J. Huang, H. Oosterhuis, M. Mansoury, H. van Hoof, and M. de Rijke. Going Beyond Popularity and Positivity Bias: Correcting for Multifactorial Bias in Recommender Systems, August 2023. Under review.

JH formulated the main research idea, conducted the experiments, and did most of the writing; HO and MdR led the discussions, offered valuable suggestions, and contributed significantly to the writing; MM and HvH helped with analyzing the results and contributed to the writing.

Chapter 4 is based on the following paper:

- J. Huang, H. Oosterhuis, M. de Rijke, and H. van Hoof. Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 190–199. ACM, September 2020.

JH formulated the main research idea, conducted the experiments, and did most of the writing; HO, MdR, and HvH led the discussions, offered valuable suggestions, and contributed significantly to the text.

Chapter 5 is based on the following paper:

- J. Huang, H. Oosterhuis, B. Cetinkaya, T. Rood, and M. de Rijke. State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2738–2748. ACM, July 2022.

JH formulated the main research idea, conducted the experiments, and did most of the writing; BC and TR helped with running the experiments; HO and MdR contributed significantly to the writing.

The writing of the thesis also benefited from work on the following publications:

- M. Li, J. Huang, and M. de Rijke. Repetition and Exploration in Offline Reinforcement Learning-based Recommendations, October 2023. DRL4IR workshop at CIKM 2023.

- X. Xin, X. Zhao, J. Huang, W. Zhang, L. Zhao, D. Yin, and G. H. Yang. DRL4IR: 4th Workshop on Deep Reinforcement Learning for Information Retrieval. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 5304–5307. ACM, October 2023.
- S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Recent Advances in the Foundations and Applications of Unbiased Learning to Rank. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3440–3443. ACM, July 2023.
- J. Huang, Z. Ren, W. X. Zhao, G. He, J.-R. Wen, and D. Dong. Taxonomy-Aware Multi-Hop Reasoning Networks for Sequential Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 573–581. ACM, February 2019.
- J. Huang, W. X. Zhao, H. Dou, J.-R. Wen, and E. Y. Chang. Improving Sequential Recommendation with Knowledge-enhanced Memory Networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 505–514. ACM, June 2018.

Part I

Correcting for Complex Forms of Selection Bias

2 Correcting for Dynamic Selection Bias

User interactions with recommender systems (RSs) are affected by user selection bias, *e.g.,* users are more likely to rate popular items (popularity bias) or items that they expect to enjoy beforehand (positivity bias). Methods exist for mitigating the effects of selection bias in user ratings on the evaluation and optimization of RSs. However, these methods treat selection bias as static, despite the fact that the popularity of an item may change drastically over time and the fact that user preferences may also change over time. In this chapter we first verify that the dynamic scenario is realistic by asking the thesis-level research question:

RQ1 Do we find evidence for dynamic selection bias and dynamic user preferences in real-world data?

We focus on the age of an item and its effect on selection bias and user preferences. Our experimental analysis reveals that the rating behavior of users on the MovieLens dataset is better captured by methods that consider effects from the age of items on bias and preferences.

We theoretically show that in a dynamic scenario in which both the selection bias and user preferences are dynamic, existing debiasing methods are no longer unbiased. To address this limitation, we answer the following thesis-level research question:

RQ2 Can the prevalent IPS-based debiasing method be extended to mitigate the effect of dynamic user selection bias and model dynamic user preferences?

In this chapter, we introduce DebiAsing in the dyNamiC scEnaRio (DANCER), a novel debiasing method that extends the inverse propensity scoring debiasing method to account for dynamic selection bias and user preferences. Our experimental results indicate that DANCER improves rating prediction performance compared to debiasing methods that incorrectly assume that selection bias is static in a dynamic scenario. To the best of our knowledge, DANCER is the first debiasing method that accounts for dynamic selection bias and user preferences in RSs.

This chapter was published as: J. Huang, H. Oosterhuis, and M. de Rijke. It Is Different When Items are Older: Debiasing Recommendations When Selection Bias and User Preferences are Dynamic. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 381–389. ACM, February 2022.

2.1 Introduction

User interactions with RSs are subject to selection bias, as a consequence of the selective behavior of users and of the fact that RSs actively restrict the items from which a user can choose [89, 98, 104, 117, 123]. A typical form of selection bias in RSs is *popularity bias*: popular items are often overrepresented in interaction logs because users are more likely to rate them [18, 104, 123]. Without correction, bias can affect user preference prediction [52, 117, 150] and lead to problems of over-specialization [3], filter bubbles [95, 102], and unfairness [21]. To correct for selection bias in interaction data from RSs, the task of *debiased recommendation* has been proposed. A widely-adopted method for this task makes use of inverse propensity scoring (IPS), a causal inference technique [59], and integrates it in the learning process of rating-prediction for recommendation [23, 52, 62, 117]. It estimates the probability of a rating to be observed in the dataset, and inversely weights ratings according to these probabilities so that in expectation each user-item pair is equally represented.

While the existing IPS-based debiasing method improves recommendations over methods that ignore the effect of bias, we identify two significant limitations. The way that IPS-based debiasing is being applied for recommendations assumes that (1) the effect of selection bias is static over time, and (2) user preferences remain unchanged as items get older. As we will show in Section 2.4, current IPS-based methods are unable to debias recommendations when the selection bias and user preferences are dynamic, *i.e.,* when they change over time.

In practice, selection bias is usually dynamic, not static [21, 61]. Typically, the popularity of an item changes with item-age [19, 61], *i.e.,* the time since its publication. Figure 2.1 shows the number of ratings that items received as they get older in the MovieLens dataset (red line).[1] On average, items receive the most attention during a short initial period of time after being published. Hence, instead of static selection bias, real-world user behavior may be better captured with *dynamic selection bias* that assumes different probabilities of observing user ratings at different item-ages. Besides selection bias, user preferences may also change over time [5, 60, 140]. In this paper, we will focus on the effect of item-age on user preferences, and thus, on capturing the change in user preferences as items become older. From Figure 2.1, it is clear that the average observed user rating varies with the item-age (blue line), despite the increased variance observed due to a decreasing number of logged interactions. We use the term *dynamic scenario* to refer to the combination of dynamic selection bias and dynamic user preferences occurring in a recommendation setting.

In this paper we first analyze real-world logged data to verify that the dynamic scenario is real: selection bias and user preferences are dynamic. The dynamic scenario poses a two-fold problem for existing IPS-based debiasing methods for RSs. First, they are not unbiased in dynamic scenarios. Second, existing methods [18, 117] for estimating static selection bias cannot be used to estimate dynamic selection bias. Hence, we propose and evaluate a debiasing method to account for dynamic selection bias and dynamic user preferences.

All in all, we make a three-fold contribution: (1) an analysis and estimation of dynamic selection bias and dynamic user preferences in the MovieLens dataset; (2) DANCER: a

[1] `https://grouplens.org/datasets/movielens/latest/`

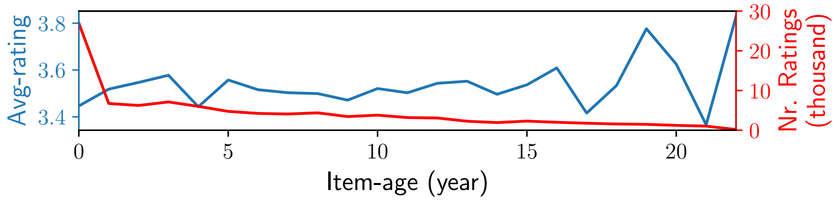


Figure 2.1: The number of ratings (indicative of popularity) and the average (observed) rating of items for different item-ages on the MovieLens-Latest-small dataset.

general debiasing method that is adaptable for DebiAsing in the dyNamiC scEnaRio; and (3) time-aware matrix factorization (TMF)-DANCER: to our knowledge, it is the first recommendation method that corrects for dynamic selection bias and models dynamic user preferences.

2.2 Related Work

General Recommendation. Early work on RSs typically uses collaborative filtering (CF) to predict user ratings on items or make recommendations to users based on the feedback of similar users with similar behavior. It is customary to divide recommendation tasks into the rating prediction task with explicit feedback (*e.g.,* user ratings) and the top-K ranking task with implicit feedback (*e.g.,* clicks). In this paper, we focus on rating prediction with explicit feedback. The traditional matrix factorization (MF) algorithm directly embeds users and items as vectors and models user-item interactions with an inner product [42, 69]. Some recent work has used deep neural networks to improve CF, *e.g.,* by using multi-layer perceptrons [27, 44], convolutional neural networks [45], or graph neural networks [46, 139]. While they significantly improve recommendation accuracy [67], they ignore the effect of time.

Time-aware Recommendation. Recently, a wide range of algorithms have been proposed that consider temporal information to improve RSs. Such methods are often classified as *time-aware* or *sequence-aware* recommendation methods. Sequence-aware recommendation methods focus on the sequential order of interactions and aim to capture a user's short-term preferences [106]. Various deep learning methods have been applied to this task [106, 161] such as recurrent neural networks [48, 142, 152], graph neural networks [143, 148], and networks with attention [25, 50, 128].

We focus on time-aware recommendation methods [17] rather than sequence-aware recommendation methods, by considering changes in user preferences over exact time periods. One of the best-known examples is time-aware matrix factorization (TMF) [68], which takes the effect of time into consideration by adding time-dependent terms to the MF model, thus allowing predicted ratings to vary over time. Koren [68] lists and compares various variants of TMF, in how well they can capture item-related or user-related temporal effects. Xiong et al. [147] propose time-aware tensor factorization (TTF): a factorization-based model that uses additional latent factors for each time period based on a probabilistic latent factor model. Lastly, the effect of time is sometimes modelled by utilizing contextual attributes related to time (*e.g.,* day of the week or season of the year) as input features for context-aware RSs [8, 17, 100, 136].

Debiased Recommendation. User selection bias is prevalent in logged data, meaning that many logged user ratings are missing not at random (MNAR) [47, 89, 117]. Two typical forms of bias in RSs are known as *popularity bias* and *positivity bias*. Popularity bias is characterized by a long tail distribution over the number of interactions per item in logged data because users are more likely to interact with more popular items [104, 123]. Positivity bias leads to an over-representation of positive feedback because users rate the items they like more often [104]. The effect of these biases is generally dynamic: they can change drastically over time [21, 61, 158]. For instance, items are rarely popular for very extended periods of time, and therefore, we may expect a dynamic effect between the age of items and popularity bias.

Existing debiasing methods for reducing the effect of selection bias address MNAR problems as follows: (1) the error-imputation-based model (EIB) fills in missing ratings with predicted values, which may introduce bias due to inaccurate predictions [122], (2) inverse propensity scoring (IPS) weights the loss associated with each observed rating inversely to their propensity, *i.e.,* the probability of observing that rating [23, 62, 117], and (3) the doubly robust (DR) method integrates the EIB and IPS approaches to overcome the high variance of IPS and the potential bias of EIB [138].

While the impact of dynamic bias has previously been pointed out [61, 158], no prior debiasing method considers a scenario in which both selection bias and user preferences change over time. All existing debiased recommendation methods assume a static effect of selection bias regardless of whether they model dynamic user preferences. Hence, there is currently no method that can effectively correct for bias in the dynamic scenario. This is the research gap that we address.

2.3 Problem Definition

We follow the common RS setting where items from the set $\mathcal{I}=\{i_1,\dots,i_M\}$ are recommended to users from the set $\mathcal{U}=\{u_1,\dots,u_N\}$ [124]. Users have preferences towards items, generally modelled by a label $y_{u,i,t}$ (*e.g.,* a rating $y_{u,i,t}\in\{1,2,3,4,5\}$) per user $u\in\mathcal{U}$ and item $i\in\mathcal{I}$. Similar to time-aware recommendations [17, 68, 147], we also consider the effect of time on user preferences: let $\mathcal{T}=\{t_1,\dots,t_T\}$ be a set of T time periods; we allow the user preference $y_{u,i,t}$ to vary over different periods $t\in\mathcal{T}$. Our goal is to optimize an RS that best captures the user preferences across all items i and time periods t. We formulate this goal as a loss function: let $\hat{y}_{u,i,t}$ be a predicted rating by the RS and $L(\hat{y},y)$ a comparison function between the predicted rating and actual rating. Then our loss is:

$$\mathcal{L}=\frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|}\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}L(\hat{y}_{u,i,t},y_{u,i,t}). \tag{2.1}$$

The function L can be chosen according to common RS metrics, for example, the prevalent mean squared error (MSE) metric:

$$L(\hat{y}_{u,i,t},y_{u,i,t})=(\hat{y}_{u,i,t}-y_{u,i,t})^2. \tag{2.2}$$

The choice for RSs to perform well across all time periods t in $\mathcal{T}$ is partially made for practical reasons; arguably, at any particular time one only needs RSs to perform well for the present and future [60]. However, in practice, data is only available about past user preferences, thus making optimization w.r.t. future preferences infeasible. Moreover, we expect that if an RS's performance generalizes well across the time periods in $\mathcal{T}$, it likely also generalizes well into the near future.

In our setting, logged interaction data is available to provide user ratings that can be used for optimization. However, it is unrealistic for all users to provide ratings for all items. In practice, user interaction data is very sparse. We will use an observation indicator matrix $\mathbf{O} \in \{0,1\}^{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|}$ that indicates what ratings are recorded in the logged interaction data and during which time period. We use $o_{u,i,t} \in \mathbf{O}$ to indicate this per rating: $o_{u,i,t} = 1$ indicates that the rating for user u on item i during time period t has been recorded in the logged data, and $o_{u,i,t} = 0$ that it is missing. The matrix $\mathbf{O}$ is strongly influenced by selection bias: certain ratings are much more likely to be observed than others. This can be due to self-selection bias: users choosing to rate certain items more often [104, 123]; or algorithmic bias: the RS used for logging choosing to show certain items more often [6, 41]. Well-known prevalent biases in RS data include: (1) popularity bias [104, 123] – often a small group of popular items receive most interactions; and (2) positivity bias [104] – users are usually more likely to rate items they prefer. We model selection bias using the probability of a rating being recorded: $p_{u,i,t} = P(o_{u,i,t} = 1)$, which we also refer to as the *observation probability* or *propensity*. Again, we deviate from the common existing method by explicitly allowing $p_{u,i,t}$ to vary over different time periods t. This enables our method to not only model a bias such as popularity bias but also how that bias changes as items get older and decline in popularity.

2.4 Estimation Ignoring Dynamic Bias

Before we introduce our recommendation method for dealing with the dynamic scenario in which both selection bias and user preferences are dynamic, we will show that, in a dynamic scenario, the existing recommendation methods that either assume no bias or static bias are not unbiased. The standard estimation of how well the predicted user preferences reflect the true user preferences shown in Eq. 2.1 is the full-information loss (*i.e.,* the loss based on all the ratings), which is impractical since user preferences are only partially known in the logged data. The naive loss ignores the effect of selection bias completely and thus assumes that the observed data represents the true user preferences unbiasedly. Under this assumption, the naive loss can be estimated by a simple average on the observed ratings:

$$\mathcal{L}_{\text{Naive}} = \frac{1}{|\{u,i,t : o_{u,i,t} = 1\}|} \sum_{u,i,t : o_{u,i,t}=1} L(\hat{y}_{u,i,t}, y_{u,i,t}). \tag{2.3}$$

And the widely-used debiasing method uses IPS estimation [59, 80] to correct for the probability that a user rates an item [117]. It uses static propensities $p_{u,i}$ that are the probability of observing a rating for item i by user u in any of the time-periods [89, 112]. These propensities ignore the dynamic aspect of selection bias, *i.e.,* that these probabilities can vary per time period t, resulting in the *static* IPS estimator: [2]

$$\mathcal{L}_{\text{staticIPS}} = \frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|} \sum_{u,i,t:o_{u,i,t}=1} \frac{L(\hat{y}_{u,i,t}, y_{u,i,t})}{p_{u,i}}. \tag{2.4}$$

Now that we have described the naive and *static* IPS-based loss functions for recommendation (that assume no bias and only static bias, respectively), we can consider the effect of dynamic selection bias.

2.4.1 Effect of Dynamic Selection Bias

Ignoring dynamic selection bias, the recommendation methods that use the naive or *static* IPS estimation are not unbiased in dynamic scenarios. To illustrate how this may happen, we use a simple example $\mathcal{X}$ with one user u, one item i and two time periods t_1 and t_2. Let y_{t_1} and y_{t_2} be the user ratings on the item at t_1 and t_2 respectively; p_{t_1} and p_{t_2} denote the probabilities of observing the ratings at t_1 and t_2, respectively. We omit the subscript of u and i if no confusion can arise. Due to dynamic user preferences and dynamic selection bias, the user ratings and observation probabilities are not constant over the different time periods: $y_{t_1} \neq y_{t_2}, \ p_{t_1} \neq p_{t_2}$. Remember that in this example the loss we wish to estimate is:

$$\mathcal{L}^{\mathcal{X}} = \frac{1}{2}(L(\hat{y}_{t_1}, y_{t_1}) + L(\hat{y}_{t_2}, y_{t_2})). \tag{2.5}$$

The expected naive loss over the observation variables becomes:

$$\mathbb{E}\left[\mathcal{L}^{\mathcal{X}}_{\text{Naive}}\right] = p_{t_1} L(\hat{y}_{t_1}, y_{t_1}) + p_{t_2} L(\hat{y}_{t_2}, y_{t_2}) - \frac{p_{t_1}p_{t_2}}{2}(L(\hat{y}_{t_1}, y_{t_1}) + L(\hat{y}_{t_2}, y_{t_2})). \tag{2.6}$$

Clearly, it is not proportional to the true loss $\mathcal{L}^{\mathcal{X}}$ when selection bias and user preferences are dynamic: if $y_{t_1} \neq y_{t_2}$ and $p_{t_1} \neq p_{t_2}$, then $\mathbb{E}\left[\mathcal{L}^{\mathcal{X}}_{\text{Naive}}\right] \not\propto \mathcal{L}^{\mathcal{X}}$. This happens because the rating with the higher probability of being observed is over-represented in the observations.

Then the *static* IPS-based debiasing method uses static propensity $p_{u,i} = p_{t_1} + (1 - p_{t_1})p_{t_2}$ that is the probability of observing a rating at time t_1 or t_2. If we consider the expected value of this estimator:

$$\mathbb{E}\left[\mathcal{L}^{\mathcal{X}}_{\text{staticIPS}}\right] = \frac{1}{2}\left(\frac{p_{t_1}}{p_{u,i}} L(\hat{y}_{t_1}, y_{t_1}) + \frac{p_{t_2}}{p_{u,i}} L(\hat{y}_{t_2}, y_{t_2})\right), \tag{2.7}$$

we see that it is not proportional to the true loss in the dynamic scenario: if $y_{t_1} \neq y_{t_2}$ and $p_{t_1} \neq p_{t_2}$, then $\mathbb{E}\left[\mathcal{L}^{\mathcal{X}}_{\text{staticIPS}}\right] \not\propto \mathcal{L}^{\mathcal{X}}$, because the *static* IPS estimation fails to address the problem that the user's rating at a time with a higher probability of being observed is more likely to be represented in logged data than at any other time. We note that the above counterexample holds regardless of whether the prediction of user ratings allows for dynamic preferences, *i.e.,* whether $\hat{y}_{t_1} = \hat{y}_{t_2}$ or $\hat{y}_{t_1} \neq \hat{y}_{t_2}$.

Our example is overly simplistic as it only contains a single user and a single item and two time periods; however, it can trivially be extended to any number of items, users or time periods. Thus, it is a significant problem for RSs that optimization with the naive or *static* IPS is not unbiased if both the user preferences and the selection bias are dynamic; it

[2] In this chapter, *static* IPS is used to highlight the IPS estimation with static propensities. This notion is particularly relevant in dynamic scenarios, where both selection bias and user preferences are dynamic.

will lead to biased optimization. Selection bias and user preferences are practically never static in the real-world; in support of this claim, Sections 2.7 and 2.8 provide evidence that the dynamic nature of bias and preferences can be observed in the MovieLens dataset.

2.5 DANCER: Debiasing Recommendations in the Dynamic Scenario

We introduce DANCER, a method for DebiAsing in the dyNamiC scEnaRio. We apply DANCER to time-aware matrix factorization (TMF), resulting in a novel rating prediction method that corrects for dynamic bias and models dynamic preferences. We introduce a propensity estimation method to estimate the probabilities of ratings being observed per time period.

2.5.1 Debiasing Recommendations

As discussed in Section 2.4, existing debiasing methods that use the naive or *static* IPS estimation are unable to debias in the dynamic scenario where selection bias and user preferences are both dynamic. As a solution, we propose DANCER. With accurate propensities $p_{u,i,t}$, dynamic selection bias can be fully corrected by applying DANCER to inversely weight the evaluation of the predicted ratings:

$$\mathcal{L}_{\text{DANCER}} = \frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|} \sum_{u,i,t:o_{u,i,t}=1} \frac{L(\hat{y}_{u,i,t}, y_{u,i,t})}{p_{u,i,t}}. \tag{2.8}$$

Unlike the naive approach $\mathcal{L}_{\text{Naive}}$ (Eq. 2.3) and the *static* IPS approach with a *static* estimator $\mathcal{L}_{\text{staticIPS}}$ (Eq. 2.4), the proposed debiasing method $\mathcal{L}_{\text{DANCER}}$ is unbiased in the dynamic scenario:

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{\text{DANCER}}] &= \frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|} \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} \frac{\mathbb{E}[o_{u,i,t}]}{p_{u,i,t}} \cdot L(\hat{y}_{u,i,t}, y_{u,i,t}) \\ &= \frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|} \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} L(\hat{y}_{u,i,t}, y_{u,i,t}) \propto \mathcal{L}. \end{aligned} \tag{2.9}$$

Because DANCER utilizes propensities that vary per time period t, it can correct for dynamic effects of bias that the existing static IPS estimators cannot. For instance, in our example $\mathcal{X}$ with a user, an item and two time periods (see Section 2.4), the expected DANCER loss becomes:

$$\mathbb{E}\left[\mathcal{L}^{\mathcal{X}}_{\text{DANCER}}\right] = \frac{1}{2}\left(p_{t_1}\frac{L(\hat{y}_{t_1}, y_{t_1})}{p_{t_1}} + p_{t_2}\frac{L(\hat{y}_{t_2}, y_{t_2})}{p_{t_2}}\right) = \mathcal{L}^{\mathcal{X}}, \tag{2.10}$$

where we can see that $\mathcal{L}^{\mathcal{X}}_{\text{DANCER}}$ is an unbiased estimation of the true loss $\mathcal{L}^{\mathcal{X}}$. Combined with a time-aware recommendation method, DANCER is able to predict that the user ratings change over time.

2.5.2 A Debiased Time-Aware Recommendation

Because we expect both selection bias and user preferences to change over time in a dynamic scenario, the rating prediction that is optimized by DANCER should also be able to account for changes in user preferences. While DANCER is not model specific, we will apply it to a time-aware matrix factorization (TMF) [68] model that accounts for temporal effects. We refer to this combination of TMF and debiasing method as TMF-DANCER. Given an observed rating $y_{u,i,t}$ from user u on item i at time t, TMF computes the predicted rating $\hat{y}_{u,i,t}$ as: $\hat{y}_{u,i,t} = \boldsymbol{p}_u^T \boldsymbol{q}_i + b_u + b_i + b + b_t$, where the $\boldsymbol{p}_u \in \mathbb{R}^d$ and $\boldsymbol{q}_i \in \mathbb{R}^d$ are embedding vectors of user u and item i, and $b_u \in \mathbb{R}$, $b_i \in \mathbb{R}$, and $b \in \mathbb{R}$ are user, item and global offsets, respectively. Crucially, b_t is a time-dependent offset and models the impact of time in rating prediction. Under this model, the proposed TMF-DANCER is optimized by minimizing the following loss:

$$\underset{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{B}}{\arg\min} \left[\sum_{u,i,t:o_{u,i,t}=1} \frac{\delta(\hat{y}_{u,i,t}, y_{u,i,t})}{p_{u,i,t}} + \lambda\big(||\boldsymbol{P}||_F^2 + ||\boldsymbol{Q}||_F^2 + ||\boldsymbol{B}||_F^2\big) \right], \tag{2.11}$$

where $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{B}$ denote the embeddings of all users, all items and all the offset terms, respectively; δ is the MSE loss function.

2.5.3 Propensity Estimation

DANCER requires accurate propensities $p_{u,i,t}$ to remove the effect of dynamic selection bias. Because it is the first method to consider dynamic selection bias in RSs, it thus also needs a novel method to estimate $p_{u,i,t} = P(o_{u,i,t} = 1)$, *i.e.,* the probability that the rating for user u and item i is observed at time t. We propose to apply a Negative Log-Likelihood (NLL) loss to the propensity estimates $\hat{p}_{u,i,t}$ and the observations made in a dataset (indicated by $o_{u,i,t}$):

$$\mathcal{L}_{\text{PE}} = \frac{1}{|\mathcal{U}| \cdot |\mathcal{I}| \cdot |\mathcal{T}|} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} L_o(\hat{p}_{u,i,t}, o_{u,i,t}), \tag{2.12}$$

where the function L_o is the NLL for each individual propensity:

$$L_o(\hat{p}_{u,i,t}, o_{u,i,t}) = o_{u,i,t} \cdot \log \hat{p}_{u,i,t} + (1 - o_{u,i,t}) \cdot \log(1 - \hat{p}_{u,i,t}). \tag{2.13}$$

Due to the large number of estimated propensities $\hat{p}_{u,i,t}$, we argue that it is best to predict them with a model. Similar to the rating prediction task, TMF and TTF [147] are potential choices to model how the propensities vary over users, items and time periods. Alternatively, one can also make simplifying assumptions in the estimations of dynamic popularity bias. For instance, $\hat{p}_{u,i,t} = \text{Pop}(i,t) := \frac{\sum_{u' \in \mathcal{U}} o_{u',i,t}}{|\mathcal{U}|}$ uses the ratio of ratings received by item i at time t. The $\text{Pop}(i,t)$ estimate is easy to compute, but it does assume that there are no differences between users when it comes to providing ratings.

Finally, we note that our proposed propensity estimation method Eq. 2.12 builds on existing methods for propensity estimation for static selection bias. Saito et al. [114] use MF instead of TMF or TTF. Similarly, the $\text{Pop}(i) := \frac{\sum_{u' \in \mathcal{U}} \sum_{t' \in \mathcal{T}} o_{u',i,t'}}{|\mathcal{U}| \cdot |\mathcal{T}|}$ is a common way to measure (static) popularity bias [18, 29, 158]. Our propensity estimation method makes these methods applicable to the dynamic scenario and enables them to provide propensities for the DANCER debiasing method.

2.6 Experiments

In our experiments, we focus on the age of an item (item-age) and the dynamic effect it has on selection bias and user preferences. From this point onwards, our notation will use t to denote how long an item has been available in the system, we will refer to this as the *age of the item*.

Because the distribution of ratings is very skewed towards young items, we divide the item-ages into seven bins whose edges are $[0,1,3,5,8,11,15,\infty]$ in years. For instance, a rating on an item when it is two-and-a-half years old will be assigned to $t=2$, and a rating when it is 15 years old will be assigned $t=7$. This can be interpreted as a specific choice for the time periods $\mathcal{T}$ and thus does not change any of the previously stated theory.

We first wish to investigate whether real-world selection bias and user preferences are affected by item-age – and are thus dynamic – and whether TMF-DANCER is more effective in a dynamic scenario than existing rating prediction methods that do not consider dynamic bias. Our experimental analysis is organized around three chapter-level research questions that refine the thesis-level research questions RQ1 and RQ2:

RQ1.1 Does item-age affect selection bias present in logged data?

RQ1.2 Does item-age affect real-world user preferences?

RQ2.1 Does the proposed TMF-DANCER method better mitigate the effect of bias in the dynamic scenario than existing debiasing methods designed for static selection bias?

To answer these questions, we make use of three different tasks based on the MovieLens-Latest-small dataset [43]. The following sections will each introduce one of these tasks and answer the corresponding research question.

All tasks use embeddings with 32 dimensions, hyperparameter tuning is applied per method and task in the following ranges: learning rate $\eta \in \{10^{-5},...,0.1\}$ and L_2 regularization weights $\lambda \in \{0,10^{-7},10^{-6},...,1.0\}$. Our implementation and hyperparameter choices are available at `https://github.com/BetsyHJ/DANCER`.

2.7 RQ1.1: Is Selection Bias Dynamic?

To answer RQ1.1: *Does item-age affect selection bias present in real-world logged data?*, we will evaluate whether methods that consider item-age can better predict which items will be rated than methods that do not. If item-age has a large effect on selection bias, it should be an essential feature for predicting whether users will rate an item.

2.7.1 Experimental Setup for RQ1.1

The goal of our first task is to predict which ratings will be observed in real-world data, in other words, across users u, items i and item-ages t the aim is to predict the observation $o_{u,i,t}$ variables. With $\hat{p}_{u,i,t}$ as the predicted probability of observation, the metrics for this task are the NLL (Eq. 2.13) and Perplexity (PPL):

$$2^{-\frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|\cdot|\mathcal{T}|}\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}o_{u,i,t}\cdot\log_2\hat{p}_{u,i,t}+(1-o_{u,i,t})\cdot\log_2(1-\hat{p}_{u,i,t})}. \tag{2.14}$$

To evaluate whether item-age has a significant effect on the observation probabilities – and thus the dynamic selection bias in the data –, we compare the performance of observation prediction methods that assume static bias with others that take item-age into account. Our comparison contains three baselines, one static method and four time-aware methods; when specifying the methods, we use σ to denote the sigmoid function, $\boldsymbol{p}_u$ for a learned user embedding, $\boldsymbol{q}_i$ for an item embedding, $\boldsymbol{a}_t$ for an embedding representing an item-age, and b_t is a learned parameter that varies per item-age t.

(1) **Constant**: The fraction of all ratings, this assumes no selection bias is present: $\hat{p}_{u,i,t} = \frac{\sum_{u' \in \mathcal{U}} \sum_{i' \in \mathcal{I}} \sum_{t' \in \mathcal{T}} o_{u',i',t'}}{|\mathcal{U}| \cdot |\mathcal{I}| \cdot |\mathcal{T}|}$.

(2) **Static Item popularity (Pop)**: The fraction of all ratings that have been given to the item; this assumes that selection bias is static over users and time: $\hat{p}_{u,i,t} = \frac{\sum_{u' \in \mathcal{U}} \sum_{t' \in \mathcal{T}} o_{u',i,t'}}{|\mathcal{U}| \cdot |\mathcal{T}|}$.

(3) **Time-aware Item Popularity (T-Pop)**: The item popularity per item-age; defined as the fraction of all ratings that have been given to item i of age t: $\hat{p}_{u,i,t} = \frac{\sum_{u' \in \mathcal{U}} o_{u',i,t}}{|\mathcal{U}|}$.

(4) **Static matrix factorization (MF)**: A standard MF model that assumes selection bias is static: $\hat{p}_{u,i,t} = \sigma(\boldsymbol{p}_u^T \boldsymbol{q}_i)$.

(5) **Time-aware matrix factorization (TMF)** [68]: TMF captures the drift in popularity as items get older by adding an age-dependent bias term: $\hat{p}_{u,i,t} = \sigma(\boldsymbol{p}_u^T \boldsymbol{q}_i + b_t)$.

(6) **Time-aware tensor factorization (TTF)** [147]: TTF extends MF by modelling the effect of item-age via element-wise multiplication: $\hat{p}_{u,i,t} = \sigma(\boldsymbol{p}_u^T (\boldsymbol{q}_i \times \boldsymbol{a}_t))$.

(7) **TTF++**: We propose a variation on TTF that models the effect via summation instead: $\hat{p}_{u,i,t} = \sigma(\boldsymbol{p}_u^T (\boldsymbol{q}_i + \boldsymbol{a}_t))$.

(8) **Time-aware matrix & tensor factorization (TMTF)**: Lastly, we propose a novel integration of TMF with TTF++: $\hat{p}_{u,i,t} = \sigma(\boldsymbol{p}_u^T (\boldsymbol{q}_i + \boldsymbol{a}_t) + b_t)$.

All models are optimized with the NLL loss as described in Section 2.5.3.

We split the dataset into training, validation and test partitions following a ratio of 7:1:2. The MovieLens-Latest-small dataset [43] consists of 100,836 ratings applied to 9,742 movies by 610 users between 1996 and 2018. We apply two splitting strategies to the data: (1) a time-based split that per user places the latest 20% of their ratings into the test set [17]; and (2) a random split that uniformly samples 20% of ratings per user. The time-based split is more realistic but makes the training and test data follow different distributions: *i.e.,* there will be more ratings on younger items in the training set than in the test set. Alternatively, the random split ensures both partitions follow the same distribution but is less realistic: *i.e.,* ratings in the test set may have taken place before ratings in the training set. For both settings, the training and validation set are uniformly randomly sampled from the data outside the test set. Since most users have an active lifecycle of less than one year, the time-based split results in a ratio between observed and missing ratings that is four times higher than the ratio in the test set; to account for this large difference in distributions we scale the predicted $\hat{p}_{u,i,t}$ by 0.25 in this setting. This leads to considerable performance improvements for all methods. Lastly, we ignore ratings outside of the user's presence in the dataset, *i.e.,* before their first rating or after their last; this prevents the methods from having to predict when users became active so that they can focus on the effect of item-age.

Table 2.1: RQ1.1 – Performance in observation prediction. Results are averages of 10 independent runs, the standard deviations are shown in brackets. † indicates a significant improvement over MF ($p < 0.01$) according to the paired-samples t-test.

Method	RANDOM		TIME-BASED	
	NLL	PPL	NLL	PPL
Constant	0.0973	1.1022	0.0337	1.0343
Pop	0.0890	1.0931	0.0404	1.0412
MF	0.0697 (0.0015)	1.0722 (0.0016)	0.0271 (0.0000)	1.0275 (0.0000)
T-Pop	0.1234	1.1314	0.0523	1.0537
TMF	0.0658† (0.0001)	1.0680† (0.0001)	**0.0267**† (0.0000)	**1.0271**† (0.0000)
TTF	0.0637† (0.0002)	1.0657† (0.0003)	0.0273 (0.0004)	1.0277 (0.0004)
TTF++	0.0632† (0.0002)	1.0653† (0.0002)	0.0268† (0.0001)	1.0271† (0.0001)
TMTF	**0.0621**† (0.0001)	**1.0641**† (0.0001)	0.0268† (0.0000)	1.0272† (0.0000)

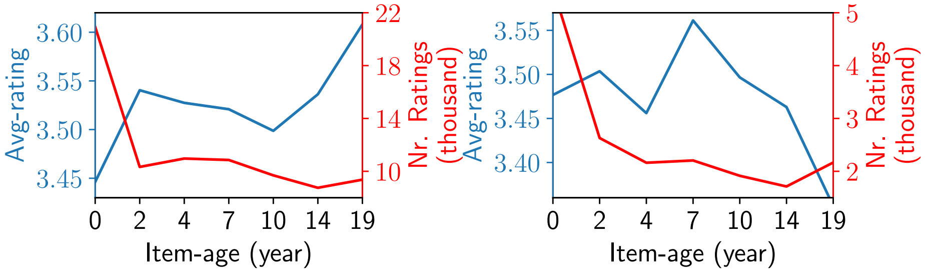


Figure 2.2: Average rating and number of ratings over item-age in the time-based partitioned training (left) and test set (right).

2.7.2 Results for RQ1.1

The results for the first task are presented in Table 2.1. Clearly, under both splitting strategies, the time-aware methods TMF, TTF++ and TMTF are significantly more accurate than Pop and MF, which assume that selection bias is static, while MF outperforms Constant, which assumes no bias. Interestingly, T-Pop performs worst among all the methods, probably due to the high variance caused by sparsity.

Under the random splitting strategy, TTF and TTF++ outperform TMF, while TMTF outperforms all other methods. Thus it appears that modelling item-age via a learned embedding better captures its effect than a single learned parameter, but moreover, TMTF shows us that combining both results in the most accurate method. Under the time-based splitting strategy, TMF performs slightly better than TTF++ and TMTF, while TTF performs worse than them. Also, Pop performs worse than Constant. A plausible reason for this inconsistency is the difference in distribution between the training and test set caused by the time-based split. The number of ratings per year displayed in Figure 2.2 displays this difference. This suggests that TMF is more robust to differences in distribution and that the other methods are somewhat overfitted on the training set. Nevertheless, most time-aware methods still predict the selection bias significantly better than the static MF.

We thus conclude that time-aware methods can better predict selection bias in real-world data than static methods. While the skewed rating distribution in Figure 2.1 already suggests that item-age has a large influence, our experimental results strongly show that item-age is an essential factor for accurately capturing the selection bias in users' rating behavior. Consequently, we answer RQ1.1 affirmatively: item-age significantly affects the selection bias present in real-world data. This result strongly implies that the assumption of static bias in previous work is incorrect, at least in recommendation settings similar to that of the MovieLens dataset.

2.8 RQ1.2: Are User Preferences Dynamic?

To answer RQ1.2: *Does item-age affect real-world user preferences?*, we compare rating prediction methods that assume preferences are static with ones that allow for dynamic preferences. If item-age has a significant effect, the latter group should perform better.

2.8.1 Experimental Setup for RQ1.2

The average rating per item-age in Figure 2.1 does not reveal a clear influence from the item-age on rating behavior. However, the averages should not be taken at face value because they are subject to selection bias. Users are generally more likely to rate movies they like (*i.e.,* positivity bias [104]), thus it is possible that while the *true* average rating drops, the *observed* remains stable due to selection bias.

To find out whether item-age has a substantial effect, we compare methods that assume static preferences with others that allow for dynamic preferences in terms of the mean squared error (MSE), mean absolute error (MAE) and Accuracy (ACC) metrics. We train and evaluate in two settings: (1) in the *observed setting* the dataset is used without any corrections to mitigate selection bias; and (2) in the *debiased setting* self-normalized inverse propensity scoring (SNIPS) [132, 149] is applied during training and metric calculation to mitigate the effect of selection bias. The advantage of the debiased setting is that – in expectation – it bases evaluation on the true rating distribution; however, it has drawbacks: it requires accurate propensities and can be subject to increased variance. The observed setting will provide biased estimates but does not have these drawbacks. Our evaluation considers both settings so that their advantages can complement each other.

The comparison includes two baselines:

(1) **Static Average Item Rating (Avg)**: The average observed rating across all item-ages: $\hat{y}_{u,i,t} = \frac{\sum_{u',i,t':o_{u',i,t'}=1} y_{u',i,t'}}{\sum_{u'\in\mathcal{U}}\sum_{t'\in\mathcal{T}} o_{u',i,t'}}$.

(2) **Time-aware Average Item Rating (T-Avg)**: the average observed rating per item-age: $\hat{y}_{u,i,t} = \frac{\sum_{u',i,t:o_{u',i,t}=1} y_{u',i,t}}{\sum_{u'\in\mathcal{U}} o_{u',i,t}}$.

In addition, we also compare with the static MF and the time-aware TMF, TTF, TTF++ and TMTF. These methods are analogous to those used in Section 2.7; the main difference is that for this task the σ sigmoid function is not applied. Additionally, we add a global offset b, a user offset b_u, and an item offset b_i to MF, TMF and TMTF. All methods are optimized to minimize MSE; in the debiased setting optimization is performed with

Table 2.2: RQ1.2 – Performance comparison of different methods in predicting ratings logged in MovieLens-Latest-small. † indicates that the improvement of the models over MF is significant ($p < 0.01$). ↑/↓ indicates whether larger or smaller values are better.

(a) In the observed setting.

Setting	Method	MSE↓	MAE↓	ACC↑
OBSERVED	Avg	0.9535	0.7540	0.2241
	MF	0.7551 (0.0046)	0.6679 (0.0021)	0.2515 (0.0016)
	T-Avg	1.0850	0.7974	0.2181
	TMF	0.7505 (0.0058)	0.6656 (0.0026)	0.2525 (0.0014)
	TTF	1.1515 (0.0542)	0.8187 (0.0181)	0.2120 (0.0054)
	TTF++	0.7526 (0.0011)	0.6645† (0.0006)	**0.2552**† (0.0007)
	TMTF	**0.7503**† (0.0014)	**0.6637**† (0.0008)	0.2533† (0.0009)

(b) In the debiased setting.

Setting	Method	SNIPS-MSE↓	SNIPS-MAE↓	SNIPS-ACC↑
DEBIASED	Avg	1.1436	0.8360	0.2048
	MF	1.2911 (0.0242)	0.8985 (0.0095)	0.1829 (0.0065)
	T-Avg	1.3105	0.8865	0.1955
	TMF	1.1210† (0.0464)	0.8383† (0.0173)	0.1944† (0.0067)
	TTF	1.8834 (0.1247)	1.0879 (0.0388)	0.1504 (0.0058)
	TTF++	1.0839† (0.0159)	0.8067† (0.0067)	**0.2134**† (0.0059)
	TMTF	**1.0727**† (0.0173)	**0.8026**† (0.0047)	0.2127† (0.0060)

DANCER following Section 2.5. We use the propensity values estimated for the previous observation prediction task by TMTF under the random-split (see Section 2.7.1).

The dataset is again partitioned into a training, validation and test set according to a ratio of 7:1:2. Unlike for the previous task (Section 2.7.1), the data for this task only consists of observed ratings, and furthermore, the partitioning is only made via uniform random sampling. As displayed in Figure 2.2, we find that a time-based split leads to extremely different rating distributions. This makes it infeasible to obtain convincing conclusions from the results of this task. Nevertheless, because a random split is perfectly suitable for evaluating a possible relationship between user preferences and item-age, our results are completely appropriate to answer RQ1.2.

2.8.2 Results for RQ1.2

Table 2.2 displays the evaluation results for the second task; in both settings the time-aware methods outperform the static MF. There is a single exception: TTF performs worst in both settings, probably due to over-fitting. The differences between the other time-aware methods and static MF are larger in the debiased setting than in the observed setting. This suggests that selection bias in the data reduces the dynamic effect of item-age on the observed ratings. We speculate that the effect of positivity bias could increase with item-age: users are less likely to try and rate movies that are older unless they already expect to enjoy them. Due to sparsity, T-Avg performs worse than Avg in both settings. Interestingly, Avg performs even better than MF in the debiased setting; this confirms prior observations that Avg is more robust in highly biased scenarios [18]. Regardless, in both settings most time-aware methods significantly outperform MF and the two baselines, and therefore, we answer RQ1.2 in the affirmative: item-age has a significant effect on user preferences.

Our conclusions for RQ1.1 and RQ1.2 indicate that the dynamic scenario, where selection bias and user preferences change over time, better captures real-world logged data, than a static view. Moreover, Section 2.4 showed that the existing *static* IPS approach cannot debias in this scenario. Consequently, our answers to RQ1.1 and RQ1.2 reveal a real need for a method that can deal with the dynamic scenario.

2.9 RQ2.1: Can TMF-DANCER Better Mitigate Dynamic Selection Bias?

Section 2.4 showed that the *static* IPS-based debiasing method is biased in a dynamic scenario. Subsequently, in Section 2.7 and 2.8 we discovered that selection bias and user preferences in the MovieLens dataset are indeed dynamic. Therefore, we can already conclude that *theoretically* TMF-DANCER is the first method that is potentially unbiased for the dynamic scenario. Our final research question considers whether this theoretical advantage translates into improved recommendation performance: RQ2.1: *Does the proposed TMF-DANCER method better mitigate the effect of bias in the dynamic scenario than existing debiasing methods designed for static selection bias?*

2.9.1 Experimental Setup for RQ2.1

The most common technique for evaluating debiasing methods for recommendation, without actual deployment to real-world users, makes use of unbiased test sets [117, 138]. This requires a dataset that has a training set consisting of biased logged ratings and a test set of user ratings on uniformly randomly selected items. Such a test set can be created by randomly sampling items and asking users to provide a rating for them, thus avoiding the selection bias that usually heavily affects what items are rated. However, the publicly available datasets that meet this criterion – YAHOO!R3 [89] and COAT SHOPPING [117] – lack any form of temporal information.[3] As a result, we cannot apply DANCER or any other form of dynamic debiasing to them.

As an alternative to using real-world datasets, we utilize a semi-synthetic simulation based on a real-world dataset for our evaluation. This simulation first estimates a simulated Ground Truth (sim-GT) based on the actual dataset and then generates a new biased training set from this sim-GT. Debiasing methods can be applied to the generated training set and evaluated on the sim-GT, since in this setting, the debiased estimates should match the sim-GT as close as possible. The creation of our semi-synthetic simulation has three steps:

[3]A recent music dataset [15] contains randomized observations and temporal information, but it only tracks user behavior during short sessions rather than for extended periods of time.

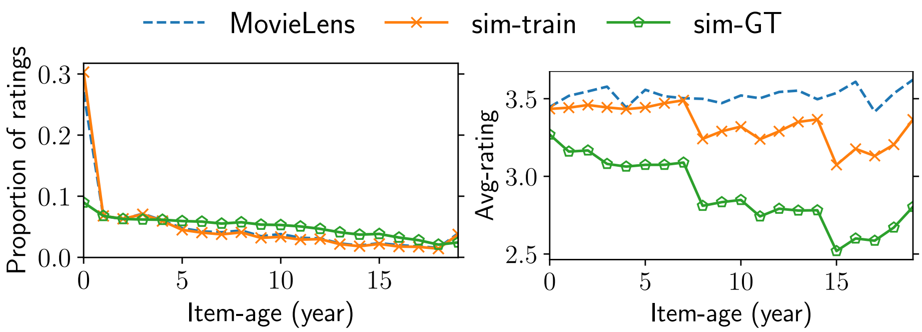


Figure 2.3: RQ2.1 – The proportion of ratings and the average rating of items over item-age on MovieLens, the simulated training set (sim-train) and the simulated Ground Truth (sim-GT).

(1) First, we estimate the complete rating matrix using the TMF method, which simply uses an age-dependent bias term to model the dynamics of user preferences, thus making the simulation understandable and not prone to overfitting. This provides us with an estimated rating for each item, user and item-age combination which we will treat as the sim-GT. By optimizing TMF with the real user ratings in the debiased setting, we hope the sim-GT reflects the real-world scenario as closely as possible.

(2) Second, dynamic selection bias is simulated using MF to model the interactions between items and item-ages. Following Section 2.7, we fit the following model: $p_{u,i,t} = \sigma(\boldsymbol{q}_i^T \boldsymbol{a}_t)$, to predict if the ratings are observed in the MovieLens dataset. To mimic real-world dynamic popularity bias more closely, we follow the user presence of the original dataset: propensities are zero before a user's first rating and after their last rating in the dataset, we also normalize the predicted probabilities so that their mean value is 4%, the same value as the dataset has.

(3) Third, to prevent overlap between the training and test set, we utilize both random and time-based splitting: per user, 50% of items are randomly selected for the test set, and a split timestep is chosen at 80% of the user presence. The test set consists of all sim-GT ratings on the randomly selected items at the last presence of each user; as a result, the test set reflects future preferences on previously unseen items. The training set uses the other 50% of items per user and samples from the ratings before the split timestamp following the estimated propensities $p_{u,i,t}$ from the previous step. The result is a training set where due to dynamic selection bias only $\sim$2% of the $y_{u,i,t}$ ratings are observed.

Figure 2.3 compares the original MovieLens dataset with our semi-synthetic simulation. The popularity of items, in terms of how many ratings they receive, is closely approximated by the simulated training set. In terms of average rating, there is some deviation between the simulated training set and MovieLens: the simulated training set rates older items lower than MovieLens. It seems likely that this is the result of positivity bias, which is not part of our simulation. Nonetheless, we clearly see that both dynamic selection bias and dynamic user preferences are represented in our simulation.

We compare the performance of TMF-DANCER with the following baselines: (1) Four

Table 2.3: RQ2.1 – Performance of TMF-DANCER compared with different methods. † indicates that the improvement of TMF-DANCER over all the baselines is significant at the level of 0.01.

Method	MSE↓		MAE↓		ACC↑	
Avg	0.3155		0.4321		0.3623	
T-Avg	0.3280		0.4326		0.3614	
MF	0.1811	(0.0030)	0.3314	(0.0028)	0.4680	(0.0040)
TMF	0.1338	(0.0019)	0.2818	(0.0022)	0.5396	(0.0038)
MF-StaticIPS	0.1879	(0.0035)	0.3377	(0.0032)	0.4598	(0.0044)
TMF-StaticIPS	0.1086	(0.0021)	0.2491	(0.0027)	0.6065	(0.0057)
MF-DANCER	0.1533	(0.0016)	0.3032	(0.0017)	0.5074	(0.0023)
TMF-DANCER	**0.1045**†	(0.0014)	**0.2444**†	(0.0018)	**0.6151**†	(0.0039)

methods that ignore bias altogether: Avg, T-Avg, MF and TMF (see Section 2.8). (2) Two methods optimized with the *static* IPS estimator: MF-staticIPS [117] and TMF-staticIPS, which use the Static Item popularity propensities from Section 2.7. (3) A static preference method with dynamic debiasing: MF-DANCER, which optimizes a (static) MF while correcting for the effect of dynamic bias.

Finally, to evaluate whether TMF-DANCER is robust to misspecified propensities, we compare its performance with using Time-Aware General Popularity (TG-Pop): $\hat{p}_{u,i,t} = \frac{\sum_{u' \in \mathcal{U}} \sum_{i' \in \mathcal{I}} o_{u',i',t}}{|\mathcal{U}| \cdot |\mathcal{I}|}$, and Time-aware Item Popularity (T-Pop) (see Section 2.7.1).

2.9.2 Results for RQ2.1

The main results of our comparison are displayed in Table 2.3. Based on the displayed results we can make four observations: (1) The average methods (Avg and T-Avg) perform considerably worse than all other methods. Clearly, matrix factorization is preferable over averaging baselines. (2) The time-based methods outperform their static counterparts by substantial margins: TMF $\succ$ MF, TMF-StaticIPS $\succ$ MF-StaticIPS, and TMF-DANCER $\succ$ MF-DANCER, except T-Avg $\prec$ Avg due to sparsity.[4] This shows that assuming static preferences can substantially hurt the performance of a method when user preferences are actually dynamic. (3) The debiased methods increase performance: MF-DANCER $\succ$ MF and TMF-DANCER $\succ$ TMF-StaticIPS $\succ$ TMF. There is a single exception: MF $\succ$ MF-staticIPS under the assumption of static bias. This surprising observation shows that DANCER is more robust to certain dynamic scenarios. (4) Finally, the best performing method is TMF-DANCER, which both models dynamic preferences and is debiased under the assumption of dynamic selection bias. While it is not a surprise that this method performs well in the scenario that it assumes, the differences with other methods are considerable and statistically significant.

In addition, Table 2.4 displays the performance of TMF-DANCER using different propensities. We see that with estimated propensities the performance of TMF-DANCER

[4]We write $A \succ B$ to indicate that method A outperforms method B.

Table 2.4: RQ2.1 – Performance of TMF-DANCER with estimated propensities and the (simulated) ground truth propensities.

Method	MSE↓	MAE↓	ACC↑
TG-Pop	0.1182 (0.0012)	0.2644 (0.0016)	0.5677 (0.0032)
T-Pop	**0.1041** (0.0015)	0.2448 (0.0022)	0.6115 (0.0055)
Ground Truth	0.1045 (0.0014)	**0.2444** (0.0018)	**0.6151** (0.0039)

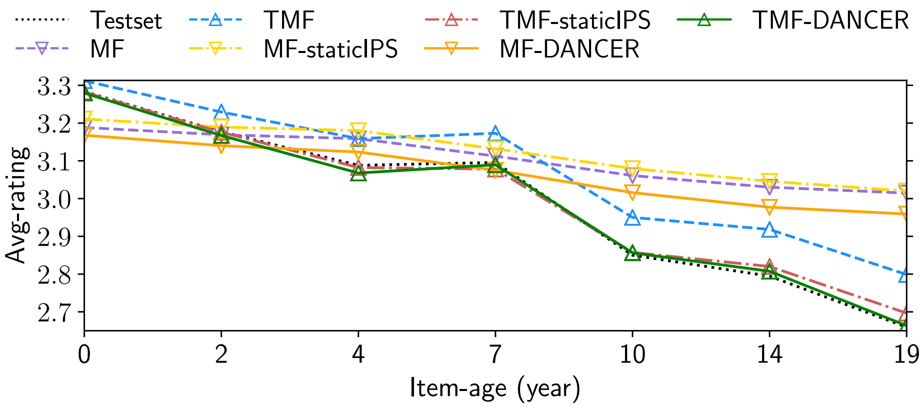


Figure 2.4: RQ2.1 – Average rating on items predicted by different models over the item-age.

is comparable to when it is using the actual sim-GT propensities. Moreover, TMF-DANCER outperforms the most baselines, except TMF-StatisIPS, even when using simple time-aware propensity estimation.

To better understand the improvements of TMF-DANCER, Figure 2.4 shows the average predicted rating from different methods across item-ages and the actual average rating. The MF methods are unable to model changes in ratings as items get older; the differences in the average ratings are purely caused by different item distributions: items that become available later in the dataset will never achieve the oldest item-ages. The TMF methods better capture the overall trend. TMF without debiasing consistently overestimates ratings; TMF-staticIPS reduces overestimation by correcting for static bias; the overestimation becomes worse for older items in both models. Instead, TMF-DANCER approximates the actual average rating at each item-age; its accuracy is quite consistent over time.

Lastly, to get more insights into the behavior of TMF-DANCER, Figure 2.5 shows the propensities and (predicted) ratings per item-age and averaged across users for two handpicked movies. We observe that TMF-DANCER outperforms TMF, especially when the popularity of items decreases as items get older.

Finally, we can answer RQ2.1 in the affirmative: the TMF-DANCER method better mitigates the effect of bias in a dynamic scenario than existing debiasing methods designed for static selection bias. This conclusion still holds when propensities are

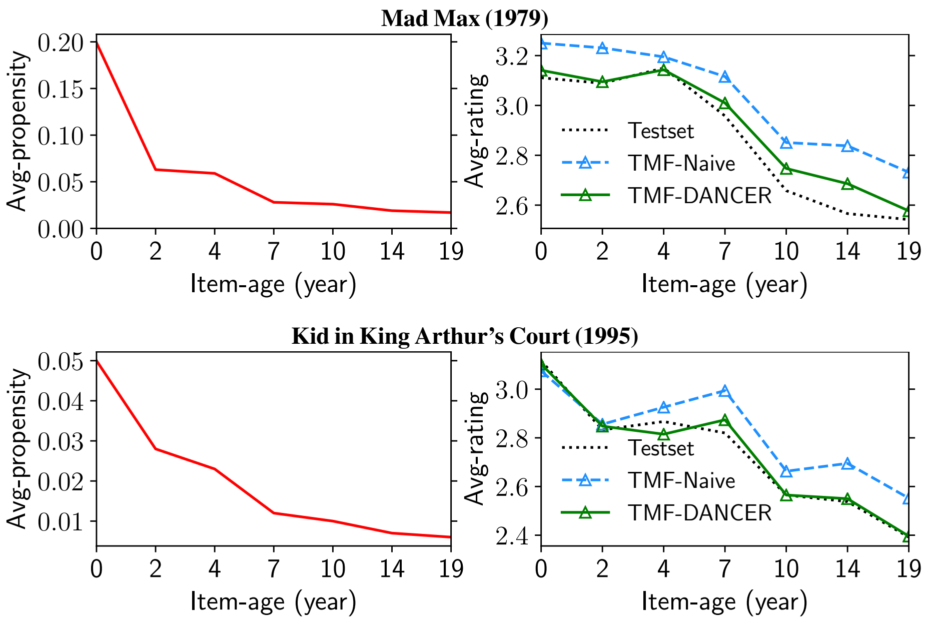


Figure 2.5: RQ2.1 – Average propensities and predicted average rating over item-age of the very popular movie “Mad Max (1979)” and the less popular “Kid in King Arthur’s Court (1995)”.

estimated, and the accuracy of TMF-DANCER is consistent across item-ages.

2.10 Conclusion

In this paper, we considered the dynamic scenario in recommendation where selection bias and user preferences change over time. Our experimental results revealed that in the real-world MovieLens dataset: (1) selection bias changes as items get older, and (2) user preferences are also affected by the age of items. Therefore, it appears that the dynamic scenario better captures the real-world situation, and thus, poses a serious problem that existing static IPS-based method cannot correct for dynamic bias in dynamic scenarios. As a solution, we proposed the DANCER debiasing method that takes into account the dynamic aspects of bias and user preferences, the first method that is unbiased in the dynamic scenario. The results on a semi-synthetic simulation based on the MovieLens dataset showed that TMF-DANCER provides significant gains in performance that are consistent across item-ages and robust to misspecified propensities. Our findings about the dynamic scenario have implications for state-of-the-art recommendation methods, as they are strongly affected by dynamic selection bias. With the DANCER debiasing method, RSs can now be expanded to deal with dynamic scenarios.

With these findings, we are now in a position to answer the thesis-level research questions RQ1 and RQ2 in the affirmative: Both selection bias and user preferences are dynamic in the real-world MovieLens dataset; furthermore, by utilizing propensities that vary per time period, a time-aware rating prediction method that is optimized by IPS can mitigate the effect of dynamic user selection bias and model dynamic user preferences.

A limitation of our work is that we only considered the rating prediction task and the effect of item-age on bias and preferences; future work should consider the ranking task and look at other aspects of time, *e.g.,* seasonal effects, weekday, time of day, etc.

3 Correcting for Multifactorial Bias

Two typical forms of bias in user interaction data with recommender systems (RSs) are popularity bias and positivity bias, which manifest themselves as the over-representation of interactions with popular items or items that users prefer, respectively. Debiasing methods aim to mitigate the effect of selection bias on the evaluation and optimization of RSs. However, existing debiasing methods only consider single-factor forms of bias, *e.g.,* only the item (popularity) or only the rating value (positivity). This is in stark contrast with the real world where user selections are generally affected by multiple factors at once.

In this chapter, we consider multifactorial selection bias in RSs. Our focus is on selection bias affected by both item and rating value factors, which is a generalization and combination of popularity and positivity bias. We address multifactorial bias by asking the following thesis-level research question:

RQ3 Can the IPS-based debiasing method be extended to correct for multifactorial bias?

We propose a propensity estimation method for multifactorial bias. By optimizing a rating prediction method with the results of our multifactorial bias propensity estimation, we correct for multifactorial bias.

While the concept of multifactorial bias is intuitive, it brings a severe practical challenge as it requires substantially more data for accurate bias estimation. This leads us to ask the following thesis-level research question in this chapter:

RQ4 Can we deal with the severe sparsity problem posed by the multifactorial method?

We propose propensity smoothing and alternating gradient descent techniques to reduce variance and improve the robustness of its optimization. Our experimental results reveal that, with our proposed techniques, multifactorial bias corrections are more effective and robust than single-factor counterparts on real-world and synthetic datasets.

3.1 Introduction

Rating prediction is a fundamental RS task where the goal is to predict user ratings on items. The task facilitates personalized recommendations to improve user satisfac-

This chapter was published as: J. Huang, H. Oosterhuis, M. Mansoury, H. van Hoof, and M. de Rijke. Going Beyond Popularity and Positivity Bias: Correcting for Multifactorial Bias in Recommender Systems, August 2023. Under review.

tion [13, 108, 109]. Rating prediction methods that are learned from user ratings can be biased as user interactions with RSs are subject to severe selection bias [89, 98, 104, 117, 123]. The effects of such bias can produce systematic errors in user preference prediction [52, 117, 150] and result in problems of over-specialization [3], filter bubbles [95, 102], and unfairness [21]. Two influential types of bias present in user rating behavior are popularity bias [18, 104, 123] and positivity bias [104], which arise as users are more likely to rate popular items or items that they prefer, respectively.

Single-factor bias. Widely-used methods for mitigating the effect of selection bias in user ratings make use of inverse propensity scoring (IPS) [59] and integrate it into the learning process [52, 62, 117]. Given the propensity of a rating, *i.e.,* the probability of the corresponding user rating the specific item, IPS weights each rating inversely to their propensity, and, thereby, corrects for the over-representation resulting from selection bias. The predominant model of popularity bias in previous work assumes that the propensity values only depend on the corresponding item. For positivity bias, the propensity values are assumed to only depend on the corresponding rating value. These single-factor propensity models can provide unbiased estimations with IPS, given that their assumptions about the factors that determine the selection bias in user data are correct. However, real-world user decisions about rating items generally depend on more than one factor, a scenario that existing methods are not designed for [34, 56, 104].

Multifactorial bias. We consider a *multifactorial* bias that is determined by two factors, *i.e.,* item and rating value. This can be seen as a generalization of popularity and positivity bias that combines the essential properties of both. As we expect multifactorial bias to better capture actual user behavior, we also expect that the resulting propensities will lead to a better performance of IPS-based debiasing methods. While this line of reasoning is intuitive, multifactorial bias also brings severe practical challenges as the consideration of multiple factors greatly increases problems of data sparsity [30, 108]. For comparison, single-factor popularity bias estimation is based on the observation frequency of ratings per item, *i.e.,* how many users have rated an item. Single-factor positivity bias estimation is based on the difference in frequency of rating values between naturally observed ratings and a (small) unbiased dataset, *i.e.,* how much more often or less often a rating value is observed in natural user interactions than when users rate randomly sampled items. Both single-factor estimation techniques already have to deal with severe sparsity, as most items are not very popular and often only very little unbiased data is available [16, 30, 108]. Multifactorial bias estimation exacerbates this sparsity problem, since it has to consider the frequencies of combinations of items and rating values. As a result, before a multifactorial bias approach can be effective, one has to first overcome this severe data-efficiency problem.

Contributions and findings. We introduce the first propensity estimation method for multifactorial bias that takes both item and rating value factors into account. To deal with the sparsity problem this poses, we propose the adoption of propensity smoothing technique and an alternating gradient descent approach for more robust and stable IPS-based optimization. Experimental results on real-world datasets show the effectiveness of our multifactorial method over state-of-the-art single-factor counterparts.

Furthermore, we perform an extensive simulation-based experimental analysis where the effect of each of the two factors is varied. The results show that single-factor methods are only effective when their corresponding factor dominates selection bias, but perform poorly when the other factor is also important. In contrast, our multifactorial approach has much more robust performance, as it is always effective, regardless of how much effect each factor has, and provides considerably better performance when both factors have a substantial effect. This indicates that, once its sparsity problem is dealt with, our multifactorial approach provides the safest choice when it is unclear what factors determine selection bias.

3.2 Preliminaries

3.2.1 Rating Prediction from User Ratings

We follow the common RS setting where users from set $\mathcal{U}=\{u_1,...,u_N\}$ give ratings on items from set $\mathcal{I}=\{i_1,...,i_M\}$ [124]. User preferences are explicitly shown by these ratings, $y_{u,i}\in\mathcal{R}=\{1,2,3,4,5\}$ per user $u\in\mathcal{U}$ and item $i\in\mathcal{I}$. Our goal is to optimize an RS model that best predicts the user ratings across all items. This is achieved by minimizing a loss function that compares the actual ratings $y_{u,i}$ and the predicted ratings $\hat{y}_{u,i}$:

$$\mathcal{L}=\frac{1}{|\mathcal{U}|\cdot|\mathcal{I}|}\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\delta(\hat{y}_{u,i},y_{u,i}), \tag{3.1}$$

where the comparison function δ can be an RS metric, *i.e.,* the commonly-used mean squared error (MSE): $\delta(\hat{y}_{u,i},y_{u,i})=(\hat{y}_{u,i}-y_{u,i})^2$.

The loss function in Eq. 3.1 naively assumes that a rating is available for each user and item. In practice, logged rating data $\mathcal{D}$ is often very sparse and subject to heavy selection bias as it is unrealistic for all users to provide ratings for all items. To indicate which ratings are available for optimization, we use an observation indicator matrix $\mathbf{O}\in\{0,1\}^{|\mathcal{U}|\cdot|\mathcal{I}|}$, where $o_{u,i}\in\mathbf{O}$ indicates whether the rating for user u on item i is recorded in the logged data ($o_{u,i}=1$) or not ($o_{u,i}=0$). One can expect $\mathbf{O}$ to be sparse and influenced by selection bias, *i.e.,* missing not at random (MNAR) [52, 117, 123]. Next, we discuss several forms that this selection bias could have in logged rating data: $\mathcal{D}=\{(u,i,y_{u,i})\,|\,u\in\mathcal{U},i\in\mathcal{I},o_{u,i}=1\}$.

3.2.2 Definition of Selection Bias

Selection bias occurs if the process that decides whether a user rates an item is not a random selection. Our definition of selection bias uses the values of propensities which are the probabilities of a user rating an item: $p_{u,i}=P(o_{u,i}=1\,|\,u,i,y_{u,i})$.

Definition 3.2.1 (Selection bias)**.** Logged rating data $\mathcal{D}$ is subject to *selection bias* if not every rating propensity has the same value:

$$\textit{Selection-bias}(\mathcal{D})\Longleftrightarrow\exists u,u'\in\mathcal{U},\exists i,i'\in\mathcal{I},p_{u,i}\neq p_{u',i'}. \tag{3.2}$$

Two influential types of user selection bias are popularity bias and positivity bias, occurring when users are more likely to rate popular items or items that they prefer, respectively. While these concepts have been widely studied in the literature [18, 104], their exact interpretation varies. We provide the following definitions of positivity bias and popularity bias to match our usage of the terms:

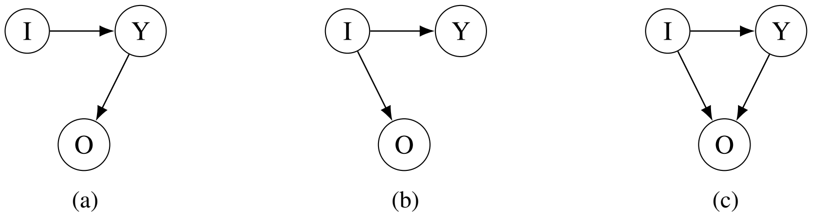


Figure 3.1: The dependency between observance (O), items (I), and rating values (Y) for different bias assumptions: (a) Positivity bias: propensities only depend on rating values; (b) Popularity bias: propensities only depend on items; (c) Multifactorial bias: propensities depend on both factors.

Definition 3.2.2 (Positivity bias)**.** Logged rating data $\mathcal{D}$ is subject to *positivity bias* if propensities only depend on their rating values (Figure 3.1a) and higher ratings correspond to higher propensities:

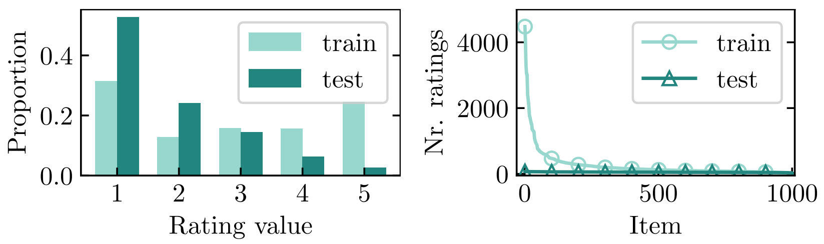
$$\begin{aligned}&\textit{Positivity-bias}(\mathcal{D}) \Longleftrightarrow \big(\textit{Selection-bias}(\mathcal{D}) \land \\ &\quad \forall u,u' \in \mathcal{U}, \forall i,i' \in \mathcal{I}, \big(y_{u,i} > y_{u',i'} \longleftrightarrow p_{u,i} > p_{u',i'}\big)\big).\end{aligned} \tag{3.3}$$

Definition 3.2.3 (Popularity bias)**.** Logged rating data $\mathcal{D}$ is subject to *popularity bias* if the propensities of ratings only depend on which item they correspond to (Figure 3.1b):

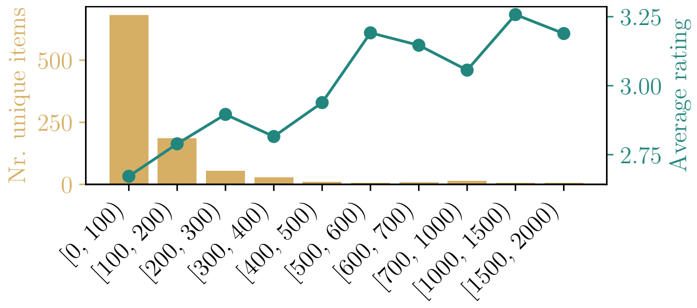
$$\begin{aligned}&\textit{Popularity-bias}(\mathcal{D}) \Longleftrightarrow \big(\textit{Selection-bias}(\mathcal{D}) \land \\ &\quad \forall u,u' \in \mathcal{U}, \forall i,i' \in \mathcal{I}, \big(i = i' \longrightarrow p_{u,i} = p_{u',i'}\big)\big).\end{aligned} \tag{3.4}$$

The definition of each form of bias exclusively focuses on the presence of its corresponding factor influences, excluding consideration of any other factors or biases. Importantly, our definitions only consider what variables the propensities of ratings depend on. Thereby, our usage of the terms is only concerned with the specific pattern the selection bias follows, not with its resulting effects. In this regard, our approach contrasts with prior work that identifies types of selection bias by the highly-skewed rating distributions that they can produce [1, 21, 104, 123]. For instance, a long-tailed rating distribution where a few items receive the most ratings (*e.g.,* Figure 3.2b) is sometimes referred to as popularity bias or evidence thereof [1, 21, 123]. Similarly, a difference between rating value frequencies from natural user behavior and ratings on randomly sampled items (*e.g.,* Figure 3.2a) is sometimes referred to as (evidence of) positivity bias [104].

However, these skewed distributions can occur for many reasons, and therefore, it is difficult to use their observation as evidence for a specific form of selection bias. For example, a long-tailed rating distribution could result from positivity bias per Definition 3.2.2: if there are only a few items with high rating values then these items will get the most ratings. Vice versa, the differences between rating distributions could result from popularity bias per Definition 3.2.3 in a case where the more popular items happen to have a higher rating on average (see Figure 3.2c). To avoid this ambiguity and since our focus is on how selection bias should be modelled, we explicitly choose to base our definitions around the dependencies of propensities and will use the terms popularity bias and positivity bias accordingly.



(a) Distribution of rating values. (b) Distributions of interactions over items.



(c) Item group by item popularity.

Figure 3.2: Skewed distributions of (a) rating values or (b) item popularity in the logged training set (train) of the Yahoo!R3 dataset, and (c) the number and average ratings of items in a group that contains items with the number of interactions falling within a certain interval are counted from logged user ratings on the self-selected songs in the Yahoo!R3 dataset.

3.3 Background on Debiasing

The loss function in Eq. 3.1 represents our ideal goal but assumes that all ratings are available, something that is rarely the case in practice. A straightforward but naive estimate of the ideal goal is to average over the observed ratings in the logged data $\mathcal{D}$:

$$\mathcal{L}_{\text{Naive}} = \frac{1}{|\mathcal{D}|} \sum_{u,i \in \mathcal{D}} \delta(\hat{y}_{u,i}, y_{u,i}). \tag{3.5}$$

However, this naive estimate ignores the effect of selection bias and assumes that every rating is equally probable to be observed [117]. As a result, if logged data $\mathcal{D}$ is subject to selection bias, it is biased by rating propensities:

$$\mathbb{E}_o[\mathcal{L}_{\text{Naive}}] = \frac{1}{|\mathcal{D}|} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} p_{u,i} \delta(\hat{y}_{u,i}, y_{u,i}) \not\propto \mathcal{L}. \tag{3.6}$$

3.3.1 IPS-based Debiasing Method

To mitigate the effect of selection bias, widely-used methods make use of inverse propensity scoring (IPS) [59] and integrate it into the learning process [52, 62, 117]. IPS weights each rating inversely to its propensity, $p_{u,i}$, and thereby, corrects for the over- and under-representation resulting from selection bias: [1]

$$\mathcal{L}_{\text{IPS}} = \frac{1}{|\mathcal{U}| \cdot |\mathcal{I}|} \sum_{u,i \in \mathcal{D}} \frac{\delta(\hat{y}_{u,i}, y_{u,i})}{p_{u,i}}. \tag{3.7}$$

Thereby, IPS gives more weight to observed ratings with small propensities and less weight to those with large propensities. Since $\mathbb{E}_o[o_{u,i}] = p_{u,i}$, the IPS loss provides an unbiased estimate of $\mathcal{L}$:

$$\mathbb{E}_o[\mathcal{L}_{\text{IPS}}] = \frac{1}{|\mathcal{U}| \cdot |\mathcal{I}|} \sum_u \sum_i \frac{\mathbb{E}_o[o_{u,i}]}{p_{u,i}} \delta(\hat{y}_{u,i}, y_{u,i}) = \mathcal{L}. \tag{3.8}$$

Combined with a recommendation method, *e.g.,* matrix factorization (MF), IPS reduces the effect of bias in predicting user ratings.

3.3.2 Existing Single-Factor Propensity Estimation

IPS for rating estimation requires propensity estimation because propensities cannot be observed directly, since the exact way users decide to rate items is not directly accessible. Methods exist for estimating propensities under our definitions of positivity bias (Definition 3.2.2) and popularity bias (Definition 3.2.3). Importantly, each existing method only corresponds to one of the definitions and thus assumes that propensities only depend on a single factor.

The predominant method of positivity bias estimation in previous work uses Bayes' rule [117]:

$$\hat{p}^{\text{pos}}_{u,i} = P(o=1 \mid y=y_{u,i}) = \frac{P(y=y_{u,i} \mid o=1) P(o=1)}{P(y=y_{u,i})}. \tag{3.9}$$

The observation prior is estimated by the observation frequency: $P(o=1) \approx |\mathcal{D}|/(|\mathcal{U}||\mathcal{I}|)$, and the conditional rating-value probability estimate is the frequency of the rating in the observed data: $P(y = r \mid o = 1) \approx \sum_{u,i \in \mathcal{D}} \mathbb{1}[y_{u,i} = r]/|\mathcal{D}|$. Finally, to estimate the rating-value prior, a small sample of unbiased (missing completely at random (MCAR)) data $\mathcal{M}$ is used; such data could be obtained by having users rate randomly sampled items. The prior estimate is simply the rating-value frequency in $\mathcal{M}$: $P(y=r) = \sum_{u,i \in \mathcal{M}} \mathbb{1}[y_{u,i} = r]/|\mathcal{M}|$. Putting these components into Eq. 3.9, we see that positivity-bias propensities are estimated as follows:

$$\hat{p}^{\text{pos}}_{u,i} = P(o=1|y=y_{u,i}) \approx \frac{|\mathcal{M}| \sum_{u',i' \in \mathcal{D}} \mathbb{1}[y_{u',i'} = y_{u,i}]}{|\mathcal{U}| \cdot |\mathcal{I}| \sum_{u',i' \in \mathcal{M}} \mathbb{1}[y_{u',i'} = y_{u,i}]}. \tag{3.10}$$

The most widely-used model of popularity bias computes propensities on items based

[1] In this and subsequent chapters, "IPS" is used in scenarios where selection bias and user preferences remain constant over time, and the distinction of *static* is not relevant. Recall in Chapter 2, we introduced "staticIPS" to emphasize the use of static propensities in dynamic scenarios to estimate IPS, as explained in that chapter.

on item popularity [114, 149]:

$$\hat{p}^{\text{pop}}_{u,i} = P(o=1 \mid i) \approx \frac{\sum_{u'} o_{u',i}}{\sum_{u'} \sum_{i'} o_{u',i'}}. \tag{3.11}$$

These estimated propensities may be small, especially for tail items, thus causing high variance in the IPS estimation. Propensity clipping is usually used as a variance reduction technique [125]; it clips propensity scores by a small value τ: $\bar{p}_{u,i} = \max(\hat{p}_{u,i}, \tau)$. Here, τ trades off the bias and variance of the IPS estimation with the clipped estimated propensities: If $\tau = 1$, it approaches the naive estimation, while if $\tau = 0$, it approaches the unbiased estimation.

With the corresponding estimated propensities, the IPS estimator can be used to mitigate the effect of popularity bias or positivity bias. However, existing single-factor forms of bias are in conflict with the fact that real-world user decisions toward rating items generally depend on more than one factor [34, 56, 104].

3.4 Correction for Multifactorial Bias

In contrast with existing single-factor models of bias, we consider a multifactorial bias that is determined by two factors: item and rating value. After defining our multifactorial bias, we introduce a stable propensity estimation method for it by adopting propensity smoothing techniques. We use IPS-based optimization with our novel estimated propensities, resulting in an unbiased rating prediction method that corrects for multifactorial bias.

3.4.1 Definition of Multifactorial Bias

Multifactorial bias occurs if the process that decides whether a user provides a rating is not a random selection and is determined by multiple factors. In this paper, we consider a specific multifactorial bias that is determined by two factors: item and rating value.

Definition 3.4.1 (Multifactorial bias)**.** Logged rating data $\mathcal{D}$ is subject to *multifactorial bias* if the propensities of ratings depend on which item they correspond to and their rating values (Figure 3.1c):

$$\begin{aligned} &\textit{Multifactorial-bias}(\mathcal{D}) \Longleftrightarrow \big(\textit{Selection-bias}(\mathcal{D}) \wedge \\ &\quad \forall u,u' \in \mathcal{U}, \forall i,i' \in \mathcal{I}, (i = i' \wedge y_{u,i} = y_{u',i'}) \longrightarrow p_{u,i} = p_{u',i'}\big). \end{aligned} \tag{3.12}$$

This definition encompasses any selection bias determined by both item and rating value factors and can naturally be extended to various types of multifactorial bias.

3.4.2 Propensity Estimate for Multifactorial Bias

A novel method is required to estimate multifactorial propensities $p_{u,i} = P(o=1 \mid y = y_{u,i}, i)$ that vary over different combinations of items and rating values. We propose to decompose the multifactorial propensity with Bayes' rule:

$$\hat{p}^{\text{mul}}_{u,i} = P(o=1 \mid y = y_{u,i}, i) = \frac{P(y = y_{u,i}, i \mid o=1) P(o=1)}{P(y = y_{u,i}, i)}, \tag{3.13}$$

and use a maximum likelihood estimate for each component. Our observation prior estimate is the observation frequency: $P(o=1)\approx|\mathcal{D}|/(|\mathcal{U}|\cdot|\mathcal{I}|)$. Our conditional joint rating-value and item probability estimate is their frequency in the observation data $\mathcal{D}$:

$$P(y=r,i\,|\,o=1)\approx\sum\nolimits_{u,i'\in\mathcal{D}}\mathbb{1}[i'=i\wedge y_{u,i'}=r]/|\mathcal{D}|, \tag{3.14}$$

and our joint rating-value and item prior estimate is their joint frequency in the small unbiased (MCAR) data $\mathcal{M}$:

$$P(y=r,i)\approx\sum\nolimits_{u,i'\in\mathcal{M}}\mathbb{1}[i'=i\wedge y_{u,i'}=r]/|\mathcal{M}|. \tag{3.15}$$

While conceptually this propensity estimation is straightforward, it brings a severe practical challenge as it relies on the frequencies of combinations of items and rating values in the sparse observation data $\mathcal{D}$ and the even sparser unbiased data $\mathcal{M}$. As a result, estimates of the joint probabilities can be extremely small or even zero, and, thereby, potentially leading to invalid propensity estimates or extremely-high-variance IPS estimates.

To address these sparsity issues, we apply Laplace smoothing [87] to both the estimations of the joint conditional probability and joint prior. The conditional joint rating-value and item probability estimate is smoothed with parameter α_1:

$$P(y=r,i\,|\,o=1)\approx\frac{\sum_{u,i'\in\mathcal{D}}\mathbb{1}[i'=i\wedge y_{u,i'}=r]+\alpha_1}{|\mathcal{D}|+\alpha_1|\mathcal{I}|\cdot|\mathcal{R}|}. \tag{3.16}$$

The estimated joint rating-value and item prior is smoothed by α_2:

$$P(y=r,i)\approx\underbrace{\frac{\sum_{u,i'\in\mathcal{M}}\mathbb{1}[y_{u,i'}=r]}{|\mathcal{M}|}}_{\text{Estimate of } P(y=r).}\cdot\underbrace{\frac{\sum_{u,i'\in\mathcal{M}}\mathbb{1}[i'=i\wedge y_{u,i'}=r]+\alpha_2}{\sum_{u,i'\in\mathcal{M}}\mathbb{1}[y_{u,i'}=r]+\alpha_2|\mathcal{I}|}}_{\text{Smoothed estimate of } P(i|y=r).}. \tag{3.17}$$

Instead of directly smoothing the joint prior $P(y=r,i)$, we decompose it into the product of the prior $P(y=r)$ and the conditional $P(i\mid y=r)$ and only smooth the latter. We found that this provided the most robust performance; most likely because item sparsity is much more extreme than rating-value sparsity.

3.4.3 A Debiasing Method for Multifactorial Bias

Using the results of our multifactorial bias propensity estimation, a rating prediction model can be optimized with IPS while accounting for multifactorial bias. Following Schnabel et al. [117], we choose inverse-propensity-scored matrix factorization (MF-IPS) as the de-biased rating prediction method. With the propensity estimates $\hat{p}^{\text{mul}}_{u,i}$, we have our multifactorial method: MF-IPS^{Mul}. It minimizes the multifactorial IPS estimate of the MSE between the predicted ratings and the actual ratings with an added L_2-regularization term:

$$\mathcal{L}_{\text{MF-IPS}^{Mul}}(\Theta)=\frac{1}{|\mathcal{D}|}\sum_{u,i\in\mathcal{D}}\frac{\delta(\hat{y}_{u,i},y_{u,i})}{\hat{p}^{\text{mul}}_{u,i}}+\lambda||\Theta||_2^2, \tag{3.18}$$

where a predicted rating is computed by a standard MF: $\hat{y}_{u,i}=\boldsymbol{p}_u^\top\boldsymbol{q}_i+a_u+b_i+c$, which is the inner-product of embedding vectors $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ for user u and item i, together with user, item and global offsets a_u,b_i and c; and the parameter set $\Theta=\{\boldsymbol{p}_u,\boldsymbol{q}_i,a_u,b_i,c\}$ includes all parameters of MF.

Algorithm 1: Our optimization method for MF-IPS^{Mul} with our alternating gradient descent approach.

```
   Input: Observed rating data: $\mathcal{D}$; estimated propensities: $\hat{p}$.
   Output: MF-IPS$^{Mul}$ parameters: $\boldsymbol{q}_u, \boldsymbol{p}_i, a_u, b_i, c$.
1  Initialize parameters $\boldsymbol{q}_u, \boldsymbol{p}_i, a_u, b_i, c$;
2  while stop condition is not reached do
       /* Epoch to update global & user embeddings and
          offsets.                                          */
3      for each batch of $(u, i, y_{u,i})$ in a random ordering of $\mathcal{D}$ do
4          Update parameters $\boldsymbol{q}_u, a_u, c$ according to Eq. 3.19;
5      end
       /* Epoch to update item embeddings and offsets.      */
6      for each batch of $(u, i, y_{u,i})$ in a random ordering of $\mathcal{D}$ do
7          Update parameters $\boldsymbol{p}_i, b_i$ according to Eq. 3.19;
8      end
9  end
```

In the optimization of our multifactorial method, we could follow common stochastic gradient descent and iteratively sample a batch of data and update parameter $\theta \in \Theta$ according to gradient of the loss function on each data batch using the Adam optimizer [65]:

$$\theta_t = \text{ADAM}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L}_{\text{MF-IPS}^{Mul}}). \tag{3.19}$$

However, we found this concurrent gradient descent method in IPS-based optimization to be unstable in experiments on real-world data (see Section 3.5). Many data batches contain widely varied propensity estimates, and due to the very low propensities under multifactorial bias, this appears to result in severe instability between updates.

An existing alternative to the concurrent gradient descent is the alternating least squares (ALS) method [133]. ALS iteratively alternates between optimizing user and item embeddings via least squares to reduce optimization instability. The alternating updates mitigate the effect of noise and outlier interactions [133]. We build on the idea of alternating gradient descent from ALS and extend it to optimize generic loss functions using the Adam optimizer. Algorithm 1 shows the procedure of optimizing MF-IPS^{Mul} with our alternating gradient descent method. It begins with parameter initialization, then updates parameters over multiple epochs according to the loss on logged user ratings $\mathcal{D}$. The optimization continues until the stop condition is reached, *e.g.,* decreasing performance on the validation set or reaching a predefined number of epochs. Importantly, in each epoch, the item-related parameters $\boldsymbol{p}_i, b_i$ (line 6–8) and other parameters $\boldsymbol{q}_u, a_u, c$ (line 3–5) are updated independently and alternately. Thereby, our optimization alternately updates a subset of parameters while keeping the remaining parameters fixed in each epoch. Our experimental results on real-world data indicate this leads to increased stability and robustness (see Section 3.5).

This completes the description of our method to mitigate the effects of multifactorial bias. It optimizes a MF model for rating predictions using IPS with multifactorial bias propensity estimation that considers both item and rating value factors. In addition, we adopt propensity smoothing and alternating gradient descent to make our multifactorial method feasible and robust in practice.

3.5 Experiments on Real-world Data

Our experimental analysis on real-world datasets aims to answer two chapter-level research questions that refine the thesis-level research questions RQ3 and RQ4:

RQ3.1 Does our proposed multifactorial method better mitigate the effect of bias in logged rating data than existing single-factor debiasing methods?

RQ4.1 How do varying smoothing parameters and our alternating gradient descent approach affect our multifactorial method?

3.5.1 Experimental Setup

Our experiments are based on two real-world datasets: Yahoo!R3 [89] and Coat [117], which are publicly available and widely used to evaluate debiasing methods. Both have a training set consisting of biased ratings and a MCAR test set of user ratings on uniformly randomly selected items. We filter the users that do not appear in the test sets to make predictions more precise, resulting in 129,179 biased ratings and 54,000 unbiased ratings of 5,400 users to 1,000 items in the Yahoo!R3 dataset, and 6,960 biased ratings and 4,640 unbiased ratings of 290 users to 300 items in the Coat dataset, respectively. The biased ratings are partitioned into a training and validation set according to a ratio of 4:1. To estimate propensities, we set aside 5% and 20% of the original test sets as the small unbiased data $\mathcal{M}$ for the Yahoo!R3 and coat datasets, respectively, which ensures at least two interactions per item for estimating the conditional joint rating-value and item distribution.

To evaluate our method, we adopt evaluation metrics widely used in previous work [117, 124, 138]: MSE, root mean square error (RMSE), and mean absolute error (MAE). We further report the average RMSE performance per user (RMSE$_U$) and item (RMSE$_I$) [91], *i.e.,* we calculate the RMSE score for each individual user/item separately and then average them.

Our comparisons consider the following prediction methods: Average Item Rating (Avg), MF, and their debiased counterparts. Avg simply predicts the average observed rating of each item: $\hat{y}_{u,i} = \frac{\sum_{u',i\in\mathcal{D}} y_{u',i}}{|\{(u',i,y_{u',i})\in\mathcal{D}\}|}$; Avg-IPS predicts the inverse-propensity weighted average: $\hat{y}_{u,i} = \sum_{u',i\in\mathcal{D}} \frac{w_{u',i}\cdot y_{u',i}}{\sum_{u'',i\in\mathcal{D}} w_{u'',i}}$, where $w_{u,i} = \frac{1}{\hat{p}_{u,i}}$. MF and MF-IPS are introduced in Section 3.4. The comparison includes the following baselines: (1) MF and Avg that ignore bias altogether; (2) Avg-IPS^{MF} and MF-IPS^{MF}, two debiased methods with propensity estimation through MF with logistic regression [54, 114, 117]; (3) MF-IPS^{Pop} with single-factor popularity bias estimation;[2] (4) Avg-IPS^{Pos} and MF-IPS^{Pos} with single-factor positivity bias estimation; and (5) Avg-IPS^{Mul} with multifactorial bias estimation. Moreover, all MF-based debiased methods are optimized by

[2]Avg-IPS with single-factor popularity bias is not included as it reduces to Avg.

two optimization methods: (1) *Concurrent* gradient descent: all parameters of methods are updated concurrently; (2) *Alternating* gradient descent: the item-related parameters and other parameters are updated alternately.

Hyperparameters used in the MF-based methods are tuned per propensity estimation in the following range: the learning rate $\eta \in \{10^{-3}, 10^{-4}, 10^{-5}\}$, the L_2 regularization weights $\lambda \in \{10^{-7}, 10^{-6}, ..., 10^{-2}\}$ and the dimension of embeddings of users and items $d \in \{16, 32, 64, 128\}$. For debiasing methods with multifactorial bias estimation, we also choose the smoothing parameters $\alpha_1, \alpha_2 \in \{1, 2, ..., 10\}$. Additionally, propensity clipping and normalization are used to reduce variance and improve the robustness of methods.

Our experimental implementation and hyperparameter choices will be released upon publication of the paper.

3.5.2 Overall Performance

Tables 3.1 and 3.2 display our main experimental results on the Yahoo!R3 and Coat datasets. We make the following four observations:

Firstly, among all the methods, Avg has the worst performance; this is expected as it provides non-personalized predictions and ignores selection bias. Accordingly, MF does model individual user preferences and outperforms Avg.

Secondly, the debiasing methods that consider the effect of bias improve the performance: Avg-IPS $\succ$ Avg and MF-IPS $\succ$ MF (except for MF-IPS^{Pop} on Yahoo!R3).[3] A strong indication of the negative effect that selection bias has on rating prediction optimization.

Thirdly, in debiasing methods, positivity bias estimation performs better than popularity bias estimation, but worse than multifactorial bias estimation: Avg-IPS^{Mul} $\succ$ Avg-IPS^{Pos}; MF-IPS^{Mul} $\succ$ MF-IPS^{Pos} $\succ$ MF-IPS^{Pop}. This suggests that positivity bias has a stronger effect than popularity bias in rating predictions. MF-based propensity estimation does not perform consistently across methods and datasets and sometimes outperforms MF-IPS^{Pos}. Nevertheless, multifactorial bias estimation provides the most robust and best overall performance; and MF-IPS^{Mul} significantly outperforms all other methods on both datasets. This strongly suggests that by considering the effect of multiple factors on selection bias, the multifactorial method can better capture and correct for bias in real-world data.

Fourthly, these performance improvements are considerably enhanced with our alternating gradient descent method, which boosts the performance of MF-IPS^{Mul} on all datasets and all metrics (with the exception of MAE on Coat). Performance gains are also seen for other methods but not as consistent as for MF-IPS^{Mul}. We speculate that due to the substantially smaller multifactorial propensities, MF-IPS^{Mul} has more variance during its optimization, and therefore, alternating gradient descent can provide a more consistent improvement here.

Overall, the best-performing method is our multifactorial debiasing method with alternating gradient descent. Therefore, we answer RQ3.1 in the affirmative: The proposed multifactorial method better mitigates the effect of bias in logged rating data than methods designed for single-factor biases.

[3]We write $A \succ B$ to indicate that method A outperforms method B.

Table 3.1: Performance comparison for predicting ratings on the Yahoo!R3 dataset. Results are means of 10 independent runs with standard deviations in brackets. Bold and underlined values indicate the highest and second-highest performance per metric. † indicates that MF-IPS^{Mul} with alternating gradient descent significantly outperforms all other methods, ‡ indicates it outperforms all other methods excluding MF-IPS^{Mul} with concurrent gradient descent (paired-samples t-test ($p < 0.01$)).

	Optimization	Method	MSE	MAE	RMSE	RMSE$_U$	RMSE$_I$
Yahoo!R3	-	Avg	2.1321	1.2671	1.4602	1.4167	1.4153
		Avg-IPS^{MF}	1.8475	1.1592	1.3592	1.3189	1.3063
		Avg-IPS^{Pos}	1.1343	0.8506	1.0650	1.0002	1.0410
		Avg-IPS^{Mul}	1.1066	0.8428	1.0520	0.9809	1.0276
	Concurrent	MF	1.8296 (0.0318)	1.1305 (0.0173)	1.3526 (0.0117)	1.2593 (0.0159)	1.3325 (0.0130)
		MF-IPS^{MF}	1.7877 (0.0297)	1.0621 (0.0024)	1.3370 (0.0111)	1.2140 (0.0050)	1.3067 (0.0109)
		MF-IPS^{Pop}	1.9432 (0.0048)	1.1425 (0.0058)	1.3940 (0.0017)	1.2783 (0.0046)	1.3711 (0.0008)
		MF-IPS^{Pos}	0.9891 (0.0013)	0.7928 (0.0079)	0.9945 (0.0006)	0.9267 (0.0048)	0.9774 (0.0015)
		MF-IPS^{Mul}	0.9812 (0.0067)	$\underline{0.7737}$ (0.0116)	0.9905 (0.0034)	$\underline{0.9128}$ (0.0046)	0.9731 (0.0027)
	Alternating	MF	1.8335 (0.0236)	1.1688 (0.0077)	1.3540 (0.0088)	1.2943 (0.0101)	1.3349 (0.0073)
		MF-IPS^{MF}	1.7143 (0.0172)	1.0616 (0.0168)	1.3093 (0.0066)	1.2009 (0.0154)	1.2821 (0.0073)
		MF-IPS^{Pop}	1.9055 (0.0196)	1.1659 (0.0077)	1.3804 (0.0071)	1.2970 (0.0095)	1.3593 (0.0063)
		MF-IPS^{Pos}	$\underline{0.9762}$ (0.0034)	0.7943 (0.0099)	$\underline{0.9880}$ (0.0017)	0.9262 (0.0077)	$\underline{0.9718}$ (0.0032)
		MF-IPS^{Mul}	**0.9629**†‡ (0.0015)	**0.7700**‡ (0.0120)	**0.9813**†‡ (0.0007)	**0.9071**‡ (0.0075)	**0.9626**†‡ (0.0025)

Table 3.2: Performance comparison for predicting ratings on the Coat dataset. Results are means of 10 independent runs with standard deviations in brackets. Bold and underlined values indicate the highest and second-highest performance per metric. † indicates that MF-IPS^{Mul} with alternating gradient descent significantly outperforms all other methods, ‡ indicates it outperforms all other methods excluding MF-IPS^{Mul} with concurrent gradient descent (paired-samples t-test $(p < 0.01)$).

	Optimization	Method	MSE	MAE	RMSE	RMSE$_U$	RMSE$_I$
Coat	-	Avg	1.6521	1.0904	1.2854	1.2521	1.2605
		Avg-IPS^{MF}	1.6300	1.0792	1.2767	1.2446	1.2505
		Avg-IPS^{Pos}	1.5324	1.0373	1.2379	1.1991	1.2149
		Avg-IPS^{Mul}	1.5056	1.0438	1.2270	1.1937	1.2082
	Concurrent	MF	1.2916 (0.0108)	0.9283 (0.0074)	1.1365 (0.0048)	1.0907 (0.0049)	1.1085 (0.0049)
		MF-IPS^{MF}	1.1597 (0.0175)	0.8687 (0.0165)	1.0769 (0.0082)	1.0366 (0.0076)	1.0512 (0.0074)
		MF-IPS^{Pop}	1.2284 (0.0142)	0.9042 (0.0115)	1.1083 (0.0064)	1.0666 (0.0066)	1.0828 (0.0066)
		MF-IPS^{Pos}	1.1728 (0.0120)	0.8708 (0.0129)	1.0830 (0.0055)	1.0395 (0.0073)	1.0576 (0.0069)
		MF-IPS^{Mul}	$\underline{1.1397}$ (0.0295)	**0.8503** (0.0199)	$\underline{1.0675}$ (0.0138)	$\underline{1.0229}$ (0.0129)	$\underline{1.0415}$ (0.0124)
	Alternating	MF	1.2040 (0.0119)	0.9034 (0.0208)	1.0973 (0.0054)	1.0599 (0.0072)	1.0755 (0.0066)
		MF-IPS^{MF}	1.1641 (0.0154)	0.8730 (0.0287)	1.0789 (0.0072)	1.0417 (0.0046)	1.0546 (0.0037)
		MF-IPS^{Pop}	1.1923 (0.0049)	0.8787 (0.0124)	1.0919 (0.0022)	1.0495 (0.0045)	1.0666 (0.0037)
		MF-IPS^{Pos}	1.1717 (0.0065)	0.8672 (0.0106)	1.0825 (0.0030)	1.0385 (0.0045)	1.0563 (0.0042)
		MF-IPS^{Mul}	**1.1020**†‡ (0.0007)	$\underline{0.8552}^{\ddagger}$ (0.0023)	**1.0498**†‡ (0.0003)	**1.0110**‡ (0.0009)	**1.0275**†‡ (0.0006)

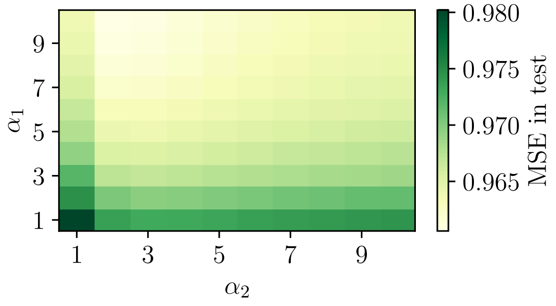


Figure 3.3: (Yahoo!R3) The effect of varying smoothing parameters α_1 and α_2 on MSE obtained by our multifactorial method.

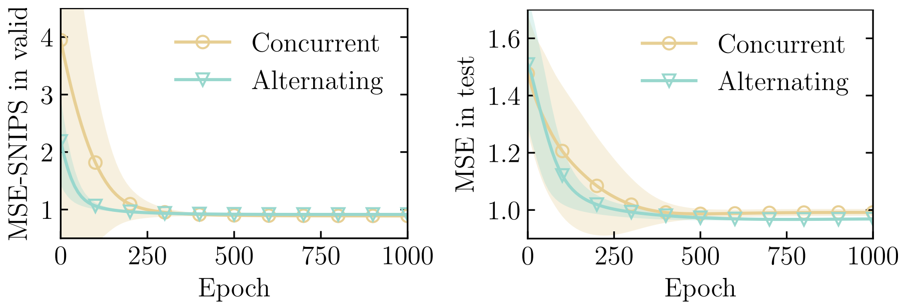


Figure 3.4: (Yahoo!R3) Learning curves tracking self-normalized IPS-weighted MSE on the validation set and MSE on the test set obtained by our multifactorial method. Results are means over 10 independent runs, shared areas indicate the 95% confident intervals.

3.5.3 Smoothing and Alternating Gradient Descent

To better understand the effect of propensity smoothing and alternating gradient descent, we perform two additional analyses. Due to space limitations, these are limited to the Yahoo!R3 dataset only.

First, we look at how the performance of our multifactorial method changes when varying the smoothing parameters. Figure 3.3 shows the MSE performance obtained for different smoothing parameters: α_1 and α_2 (see Eq. 3.16 and Eq. 3.17). We see that the highest performance is reached with $\alpha_1 = 10$ and $\alpha_2 = 2$, however, there is clearly a wide range of smoothing parameter that provide close to optimal performance. It appears that it is mainly important not to set the parameters too small, as the worst performance is reached with $\alpha_1 = 1$ and $\alpha_2 = 1$. The combined results of Figure 3.3, Table 3.1, and Table 3.2 reveal that the smoothing parameters do not need fine-tuning for the multifactorial method to outperform all other methods. Thus we conclude that propensity smoothing is an effective and robust enhancement for multifactorial debiasing.

Second, we compare the learning curves of our multifactorial method when optimization is done with the concurrent and alternating gradient descent. Figure 3.4 displays these in terms of the self-normalized IPS-weighted MSE performance [132] on the validation set and the MSE performance on the test set. Clearly, the alternating method exhibits more stable and faster learning than the concurrent method in the early stages of learning. While both converge around 500 epochs, the concurrent method converges to a slightly better MSE-IPS performance on the validation set compared to the alternating method. However, we see that this actually results in a slightly worse MSE performance on the test set, suggesting the concurrent method is more prone to overfitting. Therefore, it appears that alternating gradient descent is indeed less influenced by noise and outliers than the concurrent method, which we think is why it provides more stable and robust optimization.

Finally, we answer RQ4.1: propensity smoothing provides robust performance improvements to our multifactorial method and does not need fine-tuning; alternating gradient descent leads to less variance in learning curves and less overfitting than concurrent gradient descent. Together, these advantages substantially increase the robustness, stability and performance of our multifactorial method.

3.6 Effect of Biases on User Ratings

Our final chapter-level research question that further refines the thesis-level research question RQ4, concerns how robust our multifactorial method performs in cases where the effect of two factors on selection bias is varied:

RQ4.2 Can our multifactorial method robustly mitigate the effect of selection bias in scenarios where the effect of two factors on bias is varied?

3.6.1 Experimental Setup for RQ4.2

Due to a lack of real-world datasets with different effects of each factor, we utilize a semi-synthetic setup. We simulate a short video rating scenario by sampling user ratings on videos under different forms of selection bias. Our sampling source is the KuaiRec dataset [36] as it provides a fully observed user-item interaction matrix where 1,411 users rate almost all 3,327 items.

Since the dataset does not contain ratings but watch ratios on videos, we first convert these into 5-star user ratings. First, we sort the watch ratios in ascending order and then give the top 51.48% a rating of $y=1$, the next 25.25% get $y=2$, etc., such that the resulting ratings follow the rating distribution of the Yahoo!R3 dataset: $P(y=1)=0.5148$, $P(y=2)=0.2525$, $P(y=3)=0.1496$, $P(y=4)=0.0554$ and $P(y=5)=0.0277$.

The biased training set is constructed by sampling ratings with multifactorial selection bias. To simulate the joint effect of rating value and item factors, we first introduce two single-factor propensities: $\rho^{(\mathrm{R})}$ which is only dependent on the rating values, and $\rho^{(\mathrm{I})}$ which is only dependent on the items. Our simulated multifactorial propensity is then simply a linear interpolation between the two:

$$P(o=1\,|\,y=r,i)=\gamma\rho_r^{(\mathrm{R})}+(1-\gamma)\rho_i^{(\mathrm{I})}, \tag{3.20}$$

where $\gamma\in[0,1]$ controls the effect of each factor on the selection bias. Our simulation also covers single-factor scenarios: if $\gamma=0.0$, the selection bias is *popularity bias*, only determined by the item factor; if $\gamma=1.0$, it is *positivity bias*, only determined by the rating value factor. Importantly, when $\gamma \in (0,1)$, the resulting selection bias is multifactorial as it is affected by both factors.

Our rating-value propensities are $\rho^{(\mathrm{R})} = [0.0123, 0.0102, 0.0213, 0.0568, 0.1795]$ corresponding to the ratings $[1,2,3,4,5]$. These values were chosen to match the positivity bias propensities estimated on the Yahoo!R3 datasets, and they lead to an expectation of ratings higher than 3 being over-represented. Item propensities are generated according to a power-law distribution following Bellogín et al. [10]: $\rho^{(\mathrm{I})} = (\eta - 1) \cdot (\mathrm{rank}(i)/k_{\mathrm{min}})^{-\eta}$, where $\mathrm{rank}(i) \in [1, |\mathcal{I}|]$ is the position of item i when sorted by their average ratings descendingly, and we set the power-law exponent $\eta = 1.4$ and the minimum value $k_{\mathrm{min}} = 20$. Hereby, more popular items have a higher rating on average as is often seen in real-world data (*e.g.,* Figure 3.2c).

Some of our methods need a small unbiased MCAR set and we need an unbiased test set for evaluation. We sample unbiased data by uniform-randomly selecting 40 ratings from each user's ratings across all items. From this data, we set aside 20% for the small MCAR set and use the remaining 80% as the test set.

To answer RQ4.2, we compare the performance of our multifactorial method MF-IPS^{Mul} to that of MF with and without debiasing methods for single-factor bias correction: MF-IPS^{Pop} and MF-IPS^{Pos}. Additionally, we also consider debiasing with the ground truth propensities: MF-IPS^{GT}. This provides an unrealistic skyline that is only possible in a simulation setting where the true propensities are known. Due to space limitations, we only report MSE and MAE under optimization with alternating gradient descent.

3.6.2 Results for RQ4.2

Figure 3.5 shows the performance of the different MF with various debiasing methods, under multifactorial selection bias, as γ varies the effects of the rating-value and item factors.

We first consider when γ equals 0, and the simulated selection bias reduces to popularity bias. Here, we see that MF-IPS^{Pos} performs worst and that MF-IPS^{Mul} and MF-IPS^{Pop} have performance comparable and similar to MF. This shows that assuming selection bias is dependent on only the rating value factor can substantially hurt performance when it is actually only dependent on the item factor. However, it appears that assuming dependency on both factors does not hurt performance at all, in this scenario.

Next, we consider when γ equals 1, and the simulated selection bias reduces to positivity bias. Here, we observe that MF and MF-IPS^{Pop} perform worse than all other methods by a large margin; and that MF-IPS^{Pos} has the best performance, while our multifactorial method MF-IPS^{Mul} performs slightly worse. This strongly suggests that assuming selection bias is dependent only on the item factor is detrimental to performance when it is in fact dependent only on the rating value factor. In contrast, the multifactorial model also made an incorrect assumption: a dependency on both factors, but this only resulted in a relatively small performance decrease.

Finally, we turn our attention to all other cases: where $\gamma \in (0,1)$ and the selection bias is multifactorial bias. We see that as γ gets closer to 0 or 1, the performance of the corresponding single-factor debiasing method increases. In contrast, the performance of our multifactorial approach (MF-IPS^{Mul}) is much more stable for all values of γ, and its MSE value closely approximates that of the ground-truth method MF-IPS^{GT}. When $\gamma < 0.7$, MF-IPS^{Mul} has a substantially lower MSE than MF-IPS^{Pos}, and when $\gamma > 0.1$

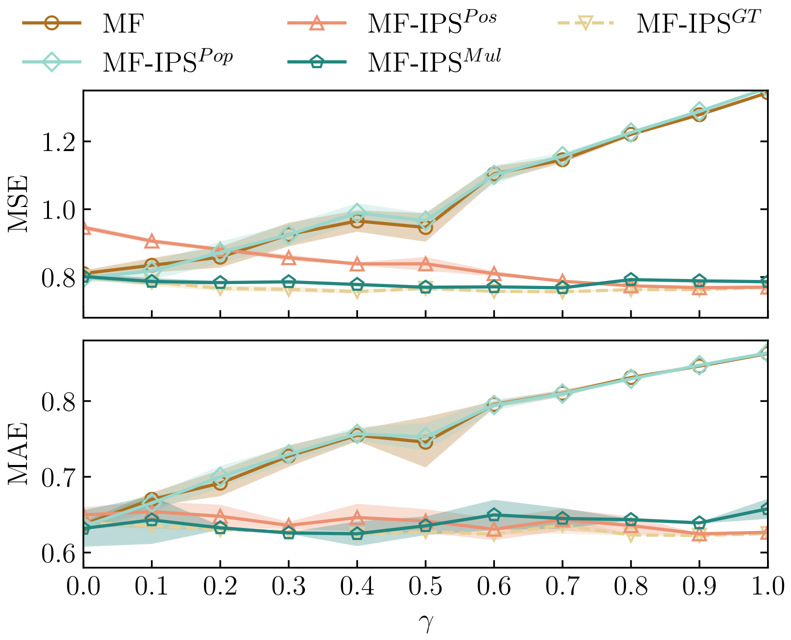


Figure 3.5: Performance in our simulated setting with different dependencies of selection bias on the item and rating value factors through varying γ (x-axis, Eq. 3.20). Results are means over 10 independent runs; shared areas show 95% confident intervals.

the MSE of MF-IPS^{Mul} is substantially lower than MF-IPS^{Pop}. There is an exception when $\gamma > 0.8$, when MF-IPS^{Mul} is outperformed by MF-IPS^{Pos} and MF-IPS^{GT} by a small but noticeable margin.

Similar observations can be made in terms of MAE performance, however, the MAE results have more variance making the trends less clearly apparent. The increased variance is likely because all methods optimize the MSE in their loss, and thus, they do not necessarily fully minimize the MAE in the process.

Overall, our results show that the performance of single-factor debiasing methods varies greatly depending on how much selection bias is affected by their corresponding factor. Conversely, the performance of our multifactorial method is hardly affected by how much selection bias depends on each factor, with only showing a minor decrease when selection bias is very close to positivity bias. Therefore, we answer RQ4.2 in the affirmative: we conclude that our multifactorial method has the most robust performance and is the safest choice if selection bias could depend on multiple factors.

3.7 Related Work

Bias is known to affect various parts of the RS pipeline, simultaneously, bias can also be amplified by RSs [21, 93]. For instance, an RS may recommend certain items to users more often, referred to as algorithmic bias [6, 41]. Furthermore, users are also more likely to interact with certain items, leading to user selection bias [88, 124]. Together, these biases result in selection bias in logged user feedback, often referred to as missing not at random (MNAR) data [47, 88, 89, 117]. They are found in both explicit feedback (*e.g.,* user ratings) [54, 117] and implicit feedback (*e.g.,* user clicks) [114, 149].

Specific types of bias in logged user ratings include popularity bias [104, 123], positivity bias [104], incentive bias [101], and conformity bias [70]. Incentive bias occurs as users are incentivized to provide rating for benefits and rewards [101]; conformity bias occurs as users tend to rate items similarly to others in a group [70]; popularity bias and positivity bias, which occur as users are more likely to rate popular items or items they prefer, have been well-studied [18, 104, 123]. Real-world data is often subject to multiple combinations of biases or complex biases which are determined by more than one factor [144, 168]. Correlations between selection and both popularity and positivity were observed in multiple real-world datasets [52, 104]. Chapter 2 also suggests that the effect of selection bias can vary over time [54]. Moreover, many contextual factors such as position, modality or surrounding items can result in bias in user rating behavior [115, 144, 169].

Debiased recommendation methods aim to mitigate the negative effects of bias and involve both bias estimation and correction [21, 117]. A prevalent family of debiasing methods is based on inverse propensity scoring (IPS) [59, 62, 117]. IPS weights observations inversely to their observation probability; in theory, its estimation is unbiased but can suffer from high variance [117]. Propensity clipping [22, 114] and doubly-robust estimation [97, 113, 138] are two common ways to reduce variance for IPS.

Bias or propensity estimation is required to estimate the probability of a user interacting with an item [92, 117]. One type of propensity estimation method is based on naive Bayes with maximum likelihood, which is commonly used for estimating popularity bias and positivity bias [18, 117, 158]. A second type of propensity estimation is based on optimizing machine learning models, for instance, logistic regression and MF models can be trained to predict propensities that can best generate an observation matrix [54, 114, 117]. While conceptually the idea of estimating propensities through optimization is appealing, our experimental results indicate that the propensity estimates are often unstable and do not always provide propensities that work well with IPS.

3.8 Conclusion

In this paper, we considered a multifactorial selection bias that is determined by two factors: item and rating value. We introduced the first propensity estimation method for multifactorial bias and integrated it into the prevalent IPS-based debiasing approach. Furthermore, we proposed the adoption of propensity smoothing and a novel alternating gradient descent method to deal with the sparsity problem that arises in multifactorial bias estimation. Our experimental results on two real-world datasets show the effectiveness of our multifactorial method over state-of-the-art single-factor counterparts. Moreover, through a simulation analysis, we found that the performance of our multifactorial method is stable as the effect of different factors is widely varied, in stark contrast with existing single-factor methods. Thereby, our multifactorial approach appears to be both substantially more robust and significantly effective than previous single-factor debiasing techniques.

With these contributions, we can answer the thesis-level research questions RQ3 and

RQ4 positively: By utilizing the results of our multifactorial bias propensity estimation, the IPS-based debiasing method can be extended to correct for multifactorial bias; moreover, the adoption of propensity smoothing techniques and an alternating gradient descent approach can effectively overcome the severe sparsity problem posed by the multifactorial method.

Our multifactorial debiasing approach could be an important contribution to the RS field, as multifactorial bias appears to better capture real-world forms of bias. Future work could extend it to implicit feedback and other recommendation settings.

Ethical Considerations

The research reported in this paper could affect applications of search and recommendation algorithms in a number of ways. Our research focus has been on bias and debiasing in recommender systems, which could reduce the problem of over-specialization, filter bubbles, and unfairness. Reducing this problem may lead to enhanced overall user experiences.

The positive societal implications of reducing the problems listed are that the associated negative societal impact, including fairness and safety considerations, can be further mitigated. While this research offers numerous societal benefits in terms of fairness, there might also be challenges associated with these efforts. It might encounter hurdles when scaling up to large-scale recommender systems due to data efficiency issues. Additionally, the challenge of distinguishing between user preferences and selection bias arises from complex user behavior.

Future research that builds on our work to improve societal outcomes could address further sources of bias that co-determine propensities so as to better capture real-world forms of bias.

Part II

Learning and Evaluating RL4Rec in a Debiased Simulator

4 A Debiased Simulator for Reinforcement Learning for Recommendation

Reinforcement learning for recommendation (RL4Rec) methods are increasingly receiving attention as an effective way to improve long-term user engagement. However, applying RL4Rec online comes with risks: exploration may lead to periods of detrimental user experience. Moreover, few researchers have access to real-world recommender systems. Simulations have been put forward as a solution where user feedback is simulated based on logged historical user data, thus enabling optimization and evaluation without being run online. While simulators do not risk the user experience and are widely accessible, we identify an important limitation of existing simulation methods. They ignore the interaction biases present in logged user data, and consequently, these biases affect the resulting simulation. In this chapter, we address this limitation by asking the thesis-level research question:

RQ5 Is it possible to mitigate the effect of bias on simulators for RL4Rec?

We introduce a debiasing step in the simulation pipeline, which corrects for the biases present in the logged data before it is used to simulate user behavior.

The existing approach to evaluate how well the simulator can generate simulated user feedback is to compare the simulated user feedback with logged user feedback. The downside of this evaluation is that it does not consider the performance of RL4Rec methods learned with this simulator, despite the fact that finding an optimal RL4Rec method is the ultimate goal. We also address this limitation by asking the thesis-level research question in this chapter:

RQ6 Can the evaluation of a simulator take the performance of the RL4Rec methods that are learned with this simulator into account?

We propose a novel evaluation approach for simulators that considers the performance of policies optimized with the simulator. Our results, based on this evaluation approach,

This chapter was published as: J. Huang, H. Oosterhuis, M. de Rijke, and H. van Hoof. Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 190–199. ACM, September 2020.

reveal that the biases from logged data negatively impact the resulting policies, unless corrected for with our debiasing method.

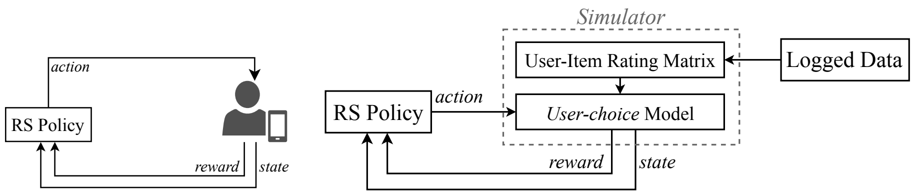
While our debiasing methods can be applied to any simulator, we make our complete pipeline publicly available as the Simulator for OFfline leArning and evaluation (SOFA): the first simulator that accounts for interaction biases prior to optimization and evaluation.

4.1 Introduction

In recent years, interest in RL4Rec has greatly increased in both academia and industry. The key idea behind reinforcement learning (RL) is to optimize a policy that matches states to actions, so that an agent performing these actions maximizes a cumulative reward [131]. RL does not just consider the immediate reward of an action, but also the effect it has on subsequent actions, allowing it to learn w.r.t. long-term goals. For recommender systems (RSs), long-term goals are usually some form of *long-term user engagement*, *e.g.,* the cumulative number of clicks or the dwell time over sessions of multiple recommendations [170]. Furthermore, RL is particularly suited for exploring the item space over multiple interactions [167], learning a recommendation policy directly from complex recommendation scenarios [24, 137, 166, 167], and adapting quickly to real-time user feedback [163]. Figure 4.1a displays the typical flow of RL4Rec: a state is the historical interactions of a user who is about to receive a recommendation, an action is an item being recommended by the policy of the recommender system, and the reward is implicit or explicit user feedback (*e.g.,* a click, a rating, dwell time, an order, etc.). The goal of RL4Rec is to maximize the cumulative reward over multiple sequential recommendations. RL methods learn from experience; in the RS setting this means that they learn by recommending items to users and observing their subsequent interactions.

Despite these advantages, the RL4Rec approach brings risks when applied online: during learning, exploratory or incorrect actions could be taken, which can be detrimental to the user experience [37, 72]. Since RL learns from experience, it is almost unavoidable that initially some disliked items are recommended. Furthermore, online deployment takes time, costs money, and many researchers – both in academia and industry – simply do not have access to an actual platform with live users.

An alternative to online experimentation is provided by *simulation-based* experiments. Here, user feedback on items is simulated in order to enable learning and evaluating RL-based RSs (see Figure 4.1b). Recently, several simulators have been proposed, specifically for RL4Rec [57, 71, 111, 118, 119, 156, 165, 167]. Some work is designed for specific datasets and specific recommendation tasks [71, 119, 156, 167], which makes them unavailable without access to similar data, and inapplicable for simulating more general recommendation tasks. For better applicability, other work has proposed simulators based on fully synthetic data which is completely generated by statistical distribution functions (*e.g.,* Bernoulli distribution) [111]. The fully synthetic approach has been criticized because it oversimplifies user behavior [118]. As a result, the resulting simulated behavior is dissimilar to the complex behavior of real users and often recognized as unrealistic feedback. To simulate user behavior while maintaining many of its natural complexities, others have proposed to simulate user feedback based on datasets of logged user data [57, 118, 165]. These simulators usually follow user preferences in the logged



(a) RL4Rec online. (b) RL4Rec with a simulator.

Figure 4.1: The general framework of RL4Rec, where a state is user historical interactions, an action is an item being recommended by the RS, and a reward is related to user feedback. (a) shows RL4Rec applied online to interact with actual users. (b) shows how RL4Rec typically interacts with a simulation-based environment.

data (*i.e.,* ratings provided by users) by basing their simulated behavior on them. E.g., a click on an item is more likely to be simulated for a user if in the logged data this user gave a high rating to this specific item. Logged data-based simulators avoid oversimplifying preferences, while still providing simulated user interactions for RL4Rec methods.

While these simulators allow for offline learning, we recognize two significant limitations: they ignore the biases present in logged data; and they have not been evaluated based on the performance of their produced policies. Biases are very prevalent in user-RS interaction data. Two influential types of biases are *popularity bias* and *positivity bias*. Popularity bias occurs because users tend to interact with more popular items [104, 123], which results in the commonly observed long tail distribution of the number of interactions per item in logged data. Positivity bias occurs because users rate the items they like more often [104], which leads to positive feedback being over-represented. These types of biases in logged data may lead to biased parameter estimation and prediction in many methods [89, 117], *e.g.,* it is known to affect matrix factorization (MF) [117]. Nevertheless, previous work on simulators has ignored biases and naively uses the observed user-item interaction data when simulating user behavior. As a result, we can expect these biases to affect the simulator and the feedback it generates. For instance, due to positivity bias, we can expect simulated users to be more positive to most items than actual users would be. Consequently, a policy learned with such a simulator would also be affected by the biases in the logged data. These biased policies may result in detrimental performance if exposed to actual users [18, 89, 117, 122]. Hence, there is a need for a simulator that is based on logged data, but that mitigates the effect of bias in the data.

Existing work has evaluated RL4Rec simulators by comparing simulated user feedback with real user feedback from logged data. For instance, some work evaluated the performance of a simulator by considering how well it predicts skip/click behaviors [165] or dwell time [166]. While this type of evaluation can simulate a single user interaction, it does not consider whether using a simulator actually leads to a well-performing policy. However, there is no work that directly considers the performance of the produced policies that result from using a simulator, despite this being the ultimate goal. E.g., if one wants to apply a policy learned in a simulator to a real-world RL4Rec setting, it is generally desired that the policy has the best performance possible. Moreover, simulators can be very effective ways to reproduce and benchmark RL4Rec methods, but such comparisons are considerably less reliable if their results are biased.

In this paper, we propose a debiasing method for RL4Rec simulators that mitigates the effect of bias in logged data. Furthermore, we introduce a novel way of evaluating the effect of bias on the final policy performance of a simulator. Our experimental results reveal that bias in logged data affects simulators and the policies they produce. While both of these contributions can be applied to any RL4Rec simulator, we combine both steps in a newly proposed SOFA. SOFA bases its simulation on a user-item rating matrix learned from logged user data; unlike existing simulators, SOFA corrects for interaction bias when learning this matrix. To evaluate SOFA, we use publicly available datasets where part of the data was logged on randomly recommended items.

The main contributions of this work are as follows:

(1) A novel approach for debiasing simulators that mitigates the effect of bias in logged data.
(2) A novel evaluation method to analyze the effect of bias on RL4Rec.
(3) Two types of experiments, both based on real-world datasets (Yahoo!R3 [89] and coat [117]) and based on a simulation study, that show that bias in logged data affects simulators and the policies they produce.
(4) SOFA, a novel simulator for RL4Rec, the first that corrects for bias in logged data.

We release the code of SOFA[1] so that future work can develop RL4Rec algorithms while mitigating the effect of bias.

4.2 Background: Reinforcement Learning for Recommendation

RL methods are commonly studied in the context of an Markov decision process (MDP), consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R}$, the transition probabilities $\mathcal{P}$, and a discount factor γ [131]. We will now describe how we model the recommendation task as an MDP [20, 22, 163, 170]:

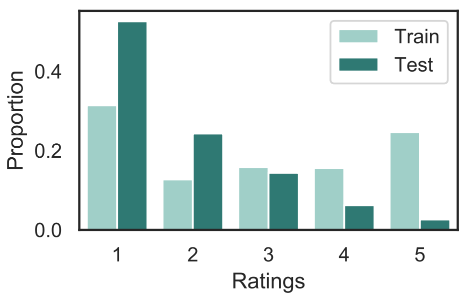
State space $\mathcal{S}$: A state represents all the current information on which a decision can be based. For RL4Rec, a state $s_t^u \in \mathcal{S}$ stores historical interactions of user u till the t-th turn of interaction, consisting of the recommended items and the corresponding feedback, denoted as $s_t^u = ([i_1,i_2,...,i_t],[f_1,f_2,...,f_t])$, with i_k the item recommended by the RS in turn k, and f_k the corresponding user feedback. The initial state $s_0^u = ([],[])$ is always empty. While contextual information about the user could be part of the state, in our experiments such information is not available.

Action space $\mathcal{A}$: Action $a_t \in \mathcal{A}$ taken by the RS consists of the recommendation of a single item i_t in turn t.

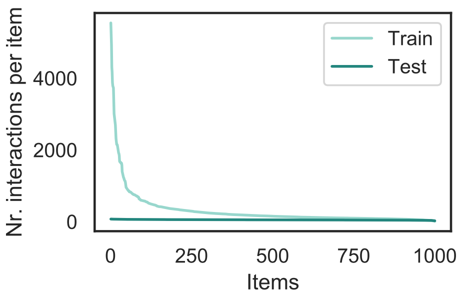
Reward $\mathcal{R}$: After receiving action a_t, consisting of item i_t being recommended by the RS, the (simulated) user gives feedback $f_t \in \{0,1\}$ (*i.e.,* skip or click) on this item. This feedback is used to generate the immediate reward $r_t = \mathcal{R}(f_t)$.

Transition probabilities $\mathcal{P}$: After the user provides feedback f_{t+1} on item i_{t+1}, the state transitions deterministically to the next state $s_{t+1}^u = ([i_1,...,i_{t+1}],[f_1,...,f_{t+1}])$. The interaction terminates after 10 turns.

[1]See `https://github.com/BetsyHJ/SOFA`.



(a) Positivity bias.



(b) Popularity bias.

Figure 4.2: Examples of positivity bias and popularity bias on the Yahoo!R3 dataset [89]. (a) shows positivity bias caused by the fact that users rate items they like more often. (b) shows popularity bias caused by the fact that users tend to interact with more popular items.

Discount factor γ: As usual in MDPs, $\gamma \in [0,1]$ aims to balance the effect of immediate rewards and future rewards. At its extremes, if $\gamma = 0$, the RS only considers the immediate reward when taking an action. When $\gamma = 1$, all future rewards will be taken into account evenly.

This completes our description of our RL4Rec MDP, which allows us to apply RL methods to recommendation. The main difference between RL4Rec and the traditional recommendation task is that RL4Rec methods: (1) make multiple sequential recommendations while keeping track of previous interactions with a user, and (2) try to optimize the cumulative rewards, based on a discounted sum on the observed user feedback f_t, the reward function $\mathcal{R}$, and the discount factor γ. Unlike the traditional recommendation setup, RL4Rec considers the long-term feedback/rewards an RS receives. We further discuss related work on RL4Rec in Section 4.4.

4.3 Background: Interaction Bias in Logged User Data

The recommendation task traditionally has a user set $\mathcal{U} = \{u_1,\ldots,u_N\}$ on the one hand and an item set $\mathcal{I} = \{i_1,\ldots,i_M\}$ on the other hand. Logged user data is usually an observed rating matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times M}$. One slot $y_{u,i}$ in this rating matrix $\boldsymbol{Y}$ denotes the rating user u would give to item i. In practice, the complete rating matrix is rarely known, since users usually do not rate every available item. We use $\mathbf{O} \in \{0,1\}^{N\times M}$ as an observation indicator: $o_{u,i} = 1$ if we observe the rating $y_{u,i}$ given by user u on item i, otherwise $o_{u,i} = 0$. In reality, observed user behavior can be affected by many types of interaction bias. Figure 4.2 visualizes the effect of positivity bias and popularity bias on the Yahoo!R3 dataset [89]. Positivity bias occurs because users rate the items they like more often [104, 123] and results in positive feedback being over-represented. In Figure 4.2a, the naturally observed ratings in the training set (Train) are compared with the test set (Test) where users were provided ratings on randomly selected items. On the randomly selected items we see only 2.6% of ratings are 5, while in the naturally logged data, the proportions of 5 are about 24.6%. Clearly, the natural user behavior results in a large "*oversampling*" of positive feedback. In contrast, popularity bias occurs because users tend to interact more with popular items [104]. Figure 4.2b shows the number of interactions per item in the logged data and reveals a clear long-tail distribution. As a result of types of bias like these, the logged data is not uniform-randomly observed, and the missing slots in the rating matrix are missing not at random (MNAR) [89].

4.3.1 Forms of Bias

Formally, MNAR is often modelled by separating the probability of observance and the probability of a rating. Generally, the rating $y_{u,i}$ a user would give is not conditioned on whether the rating is given or not:

$$P(y_{u,i}, o_{u,i}) = P(o_{u,i} \mid y_{u,i}) P(y_{u,i}). \tag{4.1}$$

We will now illustrate how this model can capture different forms of bias:

(1) **No Bias** – if every rating is equally likely to be observed, the ratings are not MNAR but missing completely at random (MCAR) and all users and items are equally represented:

$$\forall (u,u') \in \mathcal{U}, (i,i') \in \mathcal{I}, (P(o_{u,i}) = P(o_{u',i'})). \tag{4.2}$$

(2) **Positivity Bias** – when positivity bias is present, items that would receive a higher rating are more likely to be given a rating. One way to model positivity bias is to state that if an item is more preferred it is also more likely to be given a rating:

$$\forall u \in \mathcal{U}, (i,i') \in \mathcal{I}, (y_{u,i} > y_{u,i'} \rightarrow P(o_{u,i}) > P(o_{u,i'})). \tag{4.3}$$

(3) **Popularity Bias** – when popularity bias is present, items that are more popular are more likely to be given a rating. Let pop(i) denote the popularity of an item; we can model popularity bias by stating that if an item is more popular it is also more likely to be given a rating:

$$\forall u \in \mathcal{U}, (i,i') \in \mathcal{I}, (\text{pop}(i) > \text{pop}(i') \rightarrow P(o_{u,i}) > P(o_{u,i'})). \tag{4.4}$$

Now that we have described MNAR types of bias, we can consider the effect they may have on RSs and RL4Rec simulators.

4.3.2 Effect of Bias on Rating Estimation and User Simulation

Without correction, the types of interaction bias identified above will affect rating prediction, and may thus further influence the RL4Rec simulators and the policies they help produce (see Figure 4.3). To illustrate how this may happen, we will use a simple example to estimate the average rating of an item. Let avg(i) be the *true* average rating: avg$(i) = \frac{1}{N}\sum_{u \in \mathcal{U}} y_{u,i}$; the naive (uncorrected) estimate is simply the average of the observed ratings:

$$\widehat{\text{avg}}(i) = \frac{1}{\sum_{u \in \mathcal{U}} \mathbb{1}[o_{u,i} = 1]} \sum_{u \in \mathcal{U}: o_{u,i} = 1} y_{u,i}. \tag{4.5}$$

In expectation, this naive estimate is affected by the observance probabilities:

$$\mathbb{E}_o[\widehat{\text{avg}}(i)] = \frac{1}{\sum_{u \in \mathcal{U}} P(o_{u,i} = 1)} \sum_{u \in \mathcal{U}} P(o_{u,i} = 1) \cdot y_{u,i}. \tag{4.6}$$

If we compare the expected average rating estimate with the forms of bias discussed in Section 4.3.1, we see the following: (1) **No Bias** – if no bias is present (Eq. 4.2) the estimate is correct in expectation: $\mathbb{E}_o[\widehat{\text{avg}}(i)] = \text{avg}(i)$. (2) **Positivity Bias** – if positivity bias is present (Eq. 4.3), the estimate is expected to overestimate the true rating: $\mathbb{E}_o[\widehat{\text{avg}}(i)] \geq \text{avg}(i)$. This happens because higher ratings are over-represented in observance, thus the average is skewed upwards. (3) **Popularity Bias** – popularity bias (Eq. 4.4) will also affect the estimate, however, it depends on how popularity is distributed. The more popular items will have a heavier influence on the estimate, thus, if more popular items are highly rated on average, it will overestimate. Conversely, it will underestimate if more popular items are lowly rated on average.

While these effects go beyond estimating the average rating, understanding the effect of bias in this simple case helps us understand the effect it has on rating prediction. E.g., if a model is trained to predict ratings on the observed ratings, then under positivity bias we can expect it to overestimate ratings on average. For the same reasons overestimation happens on the expected average estimate: the model is trained on a sample of ratings where positive ratings were oversampled [117]. In turn, RL4Rec simulators are often based on a predicted rating matrix, and clicks are more likely to occur if an item i is recommended to a user i where a high rating $\hat{y}_{u,i}$ was predicted. Consequently, if logged data contains positivity bias, we would expect a simulator based on that data to simulate users to click more often due to the bias. In contrast, if users were asked to rate randomly sampled items, resulting in MCAR data, then we expect simulated users to click less on average. With more complicated forms of bias, such as popularity bias, the effects of the bias on the final user become less predictable. Nonetheless, without intervention we can expect bias in logged data to affect the simulated users, and unavoidably, it will thus also result in different learned RL4Rec policies. Therefore, it is important to understand the effects of interaction bias on simulations and to develop methods for mitigating them.

4.4 Related Work

RL-based Recommendation. Dulac-Arnold et al. [33] apply a Deep Deterministic Policy Gradient (DDPG) algorithm to improve the efficiency of recommender systems with a large number of items. Following this framework, Chen et al. [20] propose a tree-structured policy gradient recommendation framework, where a balanced hierarchical clustering tree is built over the items and picking an item is formulated as seeking a path from the root to a certain leaf of the tree. A branch of research has used Deep Q-Networks (DQNs) (or variants thereof) to improve recommendation performance. Zhao et al. [164] adapt a DQN architecture to incorporate positive and negative feedback of users. Others use DQN to deal with some special recommendation scenarios, such as tip recommendation [24], news recommendation [167], and recommendation mixed with advertisements [166]. Another line of research applies the Actor-Critic framework, which combines the advantages of Q-Learning and policy gradients for accelerated and stable learning. The Actor-Critic architecture is more suitable for large and dynamic action spaces and can reduce redundant computations when dealing with more complex recommendation scenarios, such as, *e.g.,* list-wise recommendation [162], page-wise recommendation [163], and dynamic treatment recommendation [137]. Choi et al. [28] use biclustering to reduce the state and action space, making the resulting MDP easy to solve with RL. Chen et al. [22] propose a policy-gradient-based algorithm that corrects for bias caused by the unobserved feedback of actions not chosen by the previous RS. Zhang et al. [155] introduce a hierarchical RL framework to improve the diversity of recommender systems.

Debiased Recommender Systems. Debiased recommendation focuses on estimating the bias (*e.g.,* positivity bias [104] and popularity bias [104, 123]) and correcting for them. Existing work on debiasing mostly focuses on missing interactions (*e.g.,* missing ratings) between users and items, and considers the case when they are missing not at random (MNAR). When missing data is missing completely at random (MCAR), maximum likelihood inference that is only based on the observed data is unbiased because of the key property of missing at random (MAR) condition that the observation process is independent of the value of unobserved data [47, 89]. In contrast, MNAR data fails to have this key property and will probably lead to biased parameter estimation and prediction because of using the incorrect likelihood function.

Methods proposed for debiasing MNAR data can be grouped into three categories. The first category applies missing data imputation on MNAR data with the joint likelihood of modeling rating prediction and the observation process [47, 89, 90]. The rating prediction model is meant to complete the rating matrix, while the observation process model is meant to learn how the data point is missing according to its value. The second category makes use of inverse propensity scoring (IPS) from causal inference [59], and integrates it in the learning process [23, 62, 117]. Based on IPS, it is able to derive an unbiased estimator for a wide range of performance estimators, such as mean squared error (MSE) and mean absolute error (MAE) used in rating prediciton models. This type of debiasing work, which separates the estimation of bias from recommendation models, makes it flexible to plug in any conditional probability estimation method as the propensity estimator [117]. The third category is a hybrid method that integrates the above two methods so as to obtain robust performance by avoiding the potentially large bias due to imputation inaccuracy and the high variance of the propensities [138].

User Simulations. A significant volume of research on RL algorithms is focused on games. As a result, many platforms have been built for learning and evaluating RL algorithms on games, such as the Arcade Learning Environment (ALE) [9]. Brockman et al. [14] collect a large series of such environments in the widely used OpenAI Gym platform [14]. An important reason for early work to consider games is that they can be simulated at scale with relatively low computational costs. Thus, RL algorithms can obtain a large number of interactions required to find the optimal policies, making research much easier.

In contrast with games, only recently simulators for RL-based RSs have been proposed. Rohde et al. [111] introduce RecoGym, which simulates an RL environment for online advertising based on completely synthetic data. However, since it uses fully synthetic data, it is unclear how well RecoGym simulates realistic user behavior. Shi et al. [118] propose PyRecGym, which bases its simulation on logged user data, and simulates a more general recommendation task. In order to aid reproducibility and sharing of models in academia, Ie et al. [57] create Recsim: a configurable simulation platform for evaluating RL-algorithms on recommendation tasks. Recsimu [165] and Virtual-Tabao [119] both use a generative adversarial network to tackle the challenges of complex item distributions based on e-commerce datasets. Surprisingly, none of the existing RL4Rec simulators

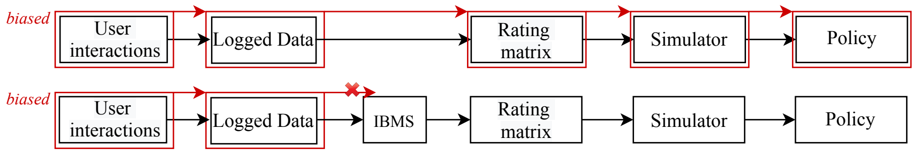


Figure 4.3: (Top): Bias, indicated in red, when present in logged user interaction data, affects all subsequent steps in simulators for RL4Rec. (Bottom): IBMS mitigates the effect of bias before it reaches the predicted rating matrix; if completely effective, IBMS prevents bias from affecting any further steps.

that are based on logged user data, consider the effect of bias in logged data.[2] Thus we can expect that bias – known to be prevalent in interaction data – affects all existing simulators, and by extension, the policies they produce. To the best of our knowledge, we are the first to consider the effects this bias may have, and whether it can be mitigated.

4.5 A Novel Method for Debiasing Simulators

We introduce a debiasing method for RL4Rec simulators and an evaluation method to measure the effect of debiasing on the policies produced by simulators. Finally, we propose the SOFA simulator, which applies both contributions.

4.5.1 Debiasing a Simulator

Ie et al. [57] define the main components of an RL4Rec simulator to be a *user model*, an *item model*, and a *user-choice model*. The *user model* and *item model* aim to capture user preference for items, while the *user-choice model* simulates user feedback when an item is recommended by an RS. Generally, user preferences are modelled using a predicted user-item rating matrix. We will focus on RL4Rec simulators that use predicted user-item ratings and a *user-choice model* on top of the predicted ratings, as shown in Figure 4.1b. We consider the case where the rating prediction model is learned from logged data.

As discussed in Section 4.3.1, logged data suffers from interaction bias, which affects any rating prediction model learned from it. Consequently, any simulator using such a prediction model will also be biased. This poses a problem for RL4Rec, since simulated user behavior should not be affected by the way a dataset was logged. As a solution, we propose the *intermediate bias mitigation step* (IBMS), an intermediate step between the logged data and the learned prediction model that aims to mitigate the bias originating from the data from affecting the model. Figure 4.3 displays where the IBMS fits in the simulator pipeline: by mitigating the effect of bias before the prediction model is learned, it minimizes its effect to reach subsequent steps, including the final produced policy.

The IBMS can apply various debiasing methods; for this paper we use the IPS approach widely used in causal inferece [59] and complete-cases analysis [80]. First, we consider

[2] Recsim [57] is an exception, as Ie et al. [57] mention bias in logged user data is a challenge but they do not propose a solution.

a standard rating prediction loss. Let $\hat{\boldsymbol{Y}}$ be the predicted ratings, $\boldsymbol{Y}$ true ratings, and $o_{u,i} = 1$ indicate that a rating from user u and item i is present in the logged data. The standard loss is based on all the pairs that are present in the logged data:

$$\mathcal{L}_{\text{Naive}} = \frac{1}{|\{(u,i): o_{u,i}=1\}|} \sum_{(u,i):o_{u,i}=1} \delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}), \tag{4.7}$$

where $\delta_{u,i}$ is chosen to match some metric, with common choices being MSE and MAE:

$$\delta^{\text{MSE}}_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = (\hat{y}_{u,i} - y_{u,i})^2, \qquad \delta^{\text{MAE}}_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = |\hat{y}_{u,i} - y_{u,i}|. \tag{4.8}$$

We call this standard loss a *naive* approach, because it assumes all ratings are equally likely to be present in the logged data, *i.e.,* the data is MCAR. In contrast, interaction data on RS is usually MNAR, which leads to a biased estimate of the full-information loss (*i.e.,* the loss based on all ratings) since:

$$\begin{aligned} E[\mathcal{L}_{\text{Naive}}] &= \frac{1}{\sum_{u=1}^{N}\sum_{i=1}^{M} P(o_{u,i}=1)} \sum_{u=1}^{N}\sum_{i=1}^{M} P(o_{u,i}=1)\delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) \\ &\neq \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}). \end{aligned} \tag{4.9}$$

Due to the effect of the bias introduced by $P(o_{u,i}=1)$, optimizing this naive loss can lead to a gross misprediction of the predicted rating matrix $\hat{\boldsymbol{Y}}$ [117, 124]. To mitigate the effect of bias in MNAR feedback, Schnabel et al. [117] apply an IPS estimator [59]. If the probabilities $P(o_{u,i}=1)$ are known, they can be corrected for by weighting the logged ratings:

$$\mathcal{L}_{\text{IPS}} = \frac{1}{N \cdot M} \sum_{(u,i):o_{u,i}=1} \frac{\delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y})}{P(o_{u,i}=1)}. \tag{4.10}$$

Basing $\mathcal{L}_{\text{IPS}}$ on logged data provides an unbiased estimate of the full-information loss:

$$E[\mathcal{L}_{\text{IPS}}] = \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \frac{P(o_{u,i}=1)\delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y})}{P(o_{u,i}=1)} = \frac{1}{N \cdot M} \sum_{u=1}^{N}\sum_{i=1}^{M} \delta_{u,i}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}). \tag{4.11}$$

For this to be truly unbiased, the exact $P(o_{u,i}=1)$ values have to be known. In practice, the logged data reveals which ratings were logged and which were not, thus an estimation method can be fitted on $o_{u,i} = 1$ to infer a model of $P(o_{u,i} = 1)$. Schnabel et al. [117] propose to use two simple propensity estimation methods: (1) naive Bayes with maximum likelihood [89], and (2) logistic regression based on features of a user-item pair [112]. By IPS weighting the ratings, the IBMS can prevent bias from affecting the rating prediction model of a simulator. In the ideal case, this removes the effect of bias on the resulting policies completely. In practice, we do not expect IBMS to completely remove bias but mitigate it to a large degree. The IBMS is applicable to any simulator that simulates interactions based on a rating prediction model.

4.5.2 Evaluating the Effect of Bias in a Simulation

To evaluate how well the IBMS mitigates bias from affecting the resulting policies, we compare the performance of a policy trained in a simulator with and without the IBMS.

Simulators are designed for situations where online deployment is impossible, thus, performance also needs to be estimated offline in these situations. Existing work has evaluated RL4Rec simulators by comparing their simulated feedback with logged user feedback [26, 119, 165], as shown in Figure 4.4a. The downside of this evaluation is that it does not consider the performance of policies learned with the simulator, despite the fact that finding an optimal policy is the ultimate goal of RL4Rec.

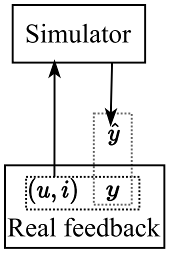
As an alternative, we propose an offline evaluation method that does consider the final produced policies. Our evaluation method only requires a sparse set of MCAR ratings, gathered on randomly selected items. Since publicly available datasets exist that meet this requirement (see Section 4.6) this method is available to all researchers in the field. Thus, we assume that a large number of MNAR ratings and a sparse set of MCAR ratings are available. Then our evaluation method consists of the following steps: (1) Train a policy using a simulator with the IBMS on the MNAR ratings, we will call the resulting policy the *debiased policy*. (2) Train another policy using an identical simulator on the MNAR ratings, expect it is does not apply IBMS, resulting in the *biased policy*. (3) Create another identical simulator, except that it is based on the MCAR ratings; call this the unbiased simulator. (4) Finally, deploy both the biased and debiased policies in the unbiased simulator to evaluate their performance by looking at cumulative reward; the difference reveals the effect of the IBMS. The intuition behind this approach to evaluation is that because MCAR data is already debiased during logging, we can create an unbiased simulator. By comparing the behavior of two policies trained with and without the IBMS in this simulator, we can see if IBMS truly removed the effect of bias. Importantly, the actual behavior of the produced policies is evaluated; this best indicates the usefulness of a simulator. While the lack of bias in MCAR data is useful, its sparsity is still a problem, as the simulator cannot simulate feedback on items without a rating. We propose two solutions, both visualized in Figure 4.4b:

Solution 1 – Limiting Action Selection: During evaluation the RS is limited to only recommend items for which ratings are available in the MCAR data. Thus for each user u the simulator finds the set of items i for which ratings $r_{u,i}$ are available in the MCAR data. The advantage of this approach is that user behavior is always based on real MCAR ratings. The disadvantage is that it limits the behavior of the RS: it could be unable to evaluate the actual behavior the RS would perform, since many items are unavailable for certain users.

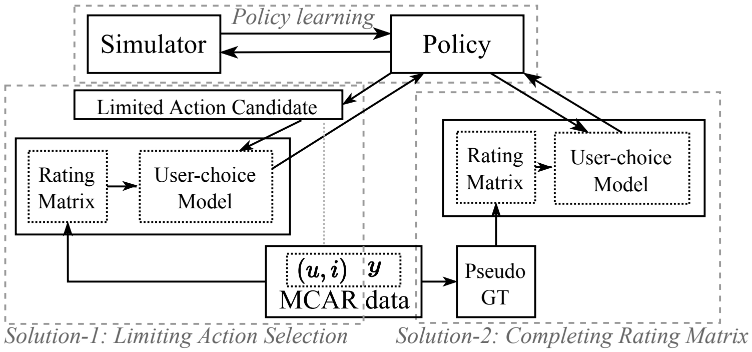
Solution 2 – Completing the Rating Matrix: To avoid limiting the behavior of the RS, a pseudo Ground Truth (GT) rating matrix could be generated using a rating prediction model learned from the MCAR data. In contrast with rating matrices based on MNAR data, the resulting pseudo GT is unbiased. The advantage is that the RS is not limited in its behavior during evaluation, thus the actual behavior it would perform is evaluated. The disadvantage is that the pseudo GT is based on predicted ratings, thus it may have some differences with the true user preferences.

4.5.3 A Simulator for Offline Learning and Evaluation

A predicted user-item rating matrix is first loaded to initialize the simulator. To simulate a user u to interact with the RS, a simulator initializes state s_0^u as empty to simulate user login. In the t-th turn of interaction, the RS recommends an item i_t as action a_t. After receiving this item, the *user-choice* model of the simulator simulates user feedback f_t on



(a) Evaluation based on observed user behavior.



(b) Evaluation with sparse MCAR data.

Figure 4.4: Different evaluation methods. (a) shows the evaluation of a simulator by comparing the simulated feedback (*e.g.,* ratings) with logged user feedback. (b) shows the process of evaluating a simulator in an unbiased simulator created from the MCAR data, where the problem caused by the sparsity of MCAR data is handled by two solutions: solution 1 is to evaluate on policy with limiting action selection shown on the left-hand side of (b), while solution 2 is to evaluate in the simulator with the complete rating matrix generated based on MCAR data shown on the right-hand side of (b).

item i_t, completes the state transition from state s_t^u to s_{t+1}^u and generates the immediate reward r_t. The RS observes feedback f_t plus the next state s_{t+1}^u, and prepares for the next turn of interaction. After K turns, the episode is terminated, and the RS saves a sequence of transitions $[(s_1,a_2,r_2,s_2),...,(s_{K-1},a_K,r_K,s_K)]$ into experience buffer $\mathcal{D}$. The transitions in $\mathcal{D}$ can be subsequently used to update the parameters of the RS.

To address the functional requirements of a simulator, we design our *Simulator for OFfline leArning and evaluation* (SOFA), a debiased simulator consisting of two components: (1) a debiased user-item rating matrix to present users' preference on items, and (2) a user-choice model to simulate user feedback, and provide the updated state and immediate reward to RS:

(1) The **debiased user-item rating matrix** is produced using the IBMS where we apply Propensity-Scored Matrix Factorization (MF-IPS) [117]. Given a user u and an item i, MF computes the predicted rating $\hat{y}_{u,i}$ as: $\hat{y}_{u,i} = \boldsymbol{p}_u{}^\top \boldsymbol{q}_i + a_u + b_i + c$, where

the $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ are embedding vectors of user u and item i, and the a_u, b_i, and c are offsets for the user, item and global respectively. MF-IPS is optimized by minimizing the prediction error between the observed ratings $y_{u,i}$ and the predicted rating $\hat{y}_{u,i}$, weighted inversely to $P(o_{u,i}=1)$:

$$\underset{\boldsymbol{P},\boldsymbol{Q},\boldsymbol{A}}{\arg\min}\left[\sum_{(u,i):o_{u,i}=1}\frac{\delta(\hat{y}_{u,i},y_{u,i})}{P(o_{u,i}=1)}+\lambda\left(||\boldsymbol{P}||_F^2+||\boldsymbol{Q}||_F^2\right)\right], \tag{4.12}$$

where $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{A}$ denote the embeddings of all users, all items, and the offset terms, respectively. Thus the final predicted rating matrix is: $\hat{\boldsymbol{Y}}=\boldsymbol{P}^\top\boldsymbol{Q}+\boldsymbol{A}$.

(2) The **user-choice model** simulates user feedback on the item being recommended from the RS, and provides the updated state and immediate reward to RS. Thus, the following steps are required for the user-choice model: (i) *Feedback simulation*: We define ratings higher than 3 as positive preference, and others as negative preference following common settings in RSs. Based on the assumption that users tend to click items if they have a positive preference for these items, if $y_{u,i_t}>3$, the user clicks item i_t, denoted as $f_t=1$; otherwise, the user skips the item, $f_t=0$. (ii) *State transition*: just concatenate i_t and f_t with s_{t-1} as the updated state s_t as defined in Section 4.2. (iii) *Reward generation*: The immediate reward r_t of *click* and *skip* feedback is specifically set to 1 and -2, these values were chosen because preliminary experiments showed they lead to efficient and stable policy learning in experiments. Finally, the user-choice model sends the updated state and the immediate reward back to the RS.

4.6 Experimental Setup

In response to the limitation of the existing simulators we point out and the solution we propose, we address three chapter-level research questions that refine the thesis-level research question RQ5 in the experiments:

RQ5.1 Does interaction bias in logged data affect a simulator?

RQ5.2 Can IBMS mitigate this bias effectively?

RQ5.3 How does the intensity of bias affect the simulators and their resulting policies?

Below, we describe the datasets and present the evaluation details for the simulator and the produced policy including the evaluation metrics and parameters used in our experiments.

Datasets. Our experiments are based on two real-world datasets and several synthetic datasets we generated ourselves, each with MNAR logged data as training set and MCAR data as test set.

Yahoo!R3 dataset [89]. The MNAR logged data of this dataset contains approximately 300,000 user-supplied ratings from 15,400 users on 1,000 items in total. The MCAR data is collected by asking 5,400 users to give ratings on 10 items randomly selected from the 1,000 items. Following [117], we consider positivity bias and use naive Bayes to estimate propensities $P(o_{u,i})$.

Coat dataset [117]. The dataset includes ratings from 290 users on 24 self-selected items and 16 randomly selected items from 300 items. Following Schnabel et al. [117], propensity $P(o_{u,i})$ is estimated by using standard regularized logistic regression trained with the profile of users (*e.g.,* gender and age) and items (*e.g.,* type and color). The bias estimated in the above way is not specified as a certain type of bias and can be recognized as a mixture of different types of biases.

Synthetic data. In order to measure the effect of the degree of bias on simulators, we generate several synthetic datasets with varying degrees of positivity bias. Unlike real-world data, this synthetic setup allows us to keep all factors constant except for the positivity bias. The generation of synthetic data involves two steps:

(1) Generate the complete true user-item rating matrix, denoted as GT. We follow Zou et al. [170] where the generation process is based on a standard Gaussian distribution. Given N users and M items, we generate the associated parameter vectors $\boldsymbol{P} \in \mathbb{R}^{N\times d}$ and $\boldsymbol{Q} \in \mathbb{R}^{M\times d}$ as profiles of users and items. $\boldsymbol{p}_u$ and $\boldsymbol{q}_i$ denoting profile vectors of user u and item i, are both drawn from the normal distribution $\mathcal{N}(0,1)$. User preference on items is determined by the inner-product of $\boldsymbol{P}$ and $\boldsymbol{Q}$, denoted as $\boldsymbol{P}^\top \boldsymbol{Q}$. GT is generated by mapping $\boldsymbol{P}^\top \boldsymbol{Q}$ into five rating bins with score from 1 to 5 according to a certain rating distribution $P(\boldsymbol{Y}=y)$. In practice, we choose $N=300$, $M=300$, $d=10$, and set $P(\boldsymbol{Y}=y)=[0.526,0.242,0.144,0.062,0.026]$ for $y=[1,2,3,4,5]$.

(2) Generate MNAR logged data under the control of observation probability:

$$P(o_{u,i} \mid y_{u,i}) = \alpha P(o_{u,i} \mid y_{u,i}, \textit{pos-bias}) + (1-\alpha) P(o_{u,i} \mid \textit{uniform}). \qquad (4.13)$$

We set the probability of uniform observation $P(o_{u,i}=1 \mid \textit{uniform}) = 5\%$ so that $\sim$5% of the user-item rating matrix is observed; thus, the remaining $\sim$95% is missing. We set $P(o_{u,i}=1 | y_{u,i}=y, \textit{pos-bias}) = [0.029,0.021,0.035,0.161,0.577]$ for $y=[1,2,3,4,5]$ to obtain a rating distribution similar to that of the Yahoo!R3 dataset. The intensity of bias is controlled by α: if $\alpha = 1.0$, the sampling probability is determined by positivity bias; if $\alpha = 0$, the logged data is sampled completely at random. We vary $\alpha \in \{0.0,0.2,0.4,0.6,0.8,1.0\}$ to generate MNAR logged data with different degrees of positivity bias.

Hyperparameters. The simulators rely on the user-item rating matrix generated by MF, including MF-IPS and MF-Naive. We followed the procedure of Schnabel et al. [117] to tune the MF hyperparameters: the L_2 regularization weight $\lambda \in \{10^{-6},...,1\}$ and dimension of embeddings of users and items $d \in \{5,10,20,40\}$, were chosen by cross-validation while considering to match the rating distributions of the predicted ratings with the real rating distributions simultaneously.[3]

For the policy used in the experiments, we use a basic DQN policy with a gated recurrent unit (GRU)-based network to encode discrete state and approximate action-value function. Due to space limitations, a detailed description of the architecture of this DQN policy is provided in the released code. The required hyperparameters come in two kinds: (1) Hyperparameters of the used DQN, *e.g.,* γ discount factor, and the dimension h of the look-up layer, and the dimension h^{GRU} of the GRU hidden state. (2) Hyperparameters used in learning process, *e.g.,* the size of replay buffer $\mathcal{D}$, the speed of greedy epsilon decay, the size of minibatch and the frequency of target network update. Following Zou et al. [170], we fix the discount factor γ to 0.9, and choose the other hyperparameters by

[3]This is slightly different from the setting in [117] without matching the real rating distributions. The median rating yields the minimal MSE loss when prediction error is large. This may cause most of the simulated feedback to be negative, and policies cannot learn useful information from the interactions.

Table 4.1: MAE, MSE, ACC, and Click-ACC performance of MF-IPS and MF-Naive compared with unbiased testset. Click-ACC means the accuracy for the click or skip behaviors generated from the rating scores. $\downarrow / \uparrow$ indicate smaller is better or worse.

Dataset	Method	MSE$\downarrow$	MAE$\downarrow$	ACC$\uparrow$	Click-ACC$\uparrow$
Yahoo!R3	MF-IPS	1.518	0.999	0.336	0.889
	MF-Naive	2.263	1.287	0.222	0.761
Coat	MF-IPS	1.129	0.878	0.311	0.827
	MF-Naive	1.217	0.914	0.287	0.830
Synthetic	MF-IPS	1.445	0.997	0.284	0.901
	MF-Naive	1.780	1.093	0.273	0.856

running multiple experiments and seeing which resulted in the most stable learning curves which was measured by the average cumulative number of clicks over 10-turn interactions with given simulators. The specific values of the hyperparameters for different datasets have been released with the code.

Evaluation Metrics. To evaluate the performance of a policy, we use the cumulative number of clicks received over 10 interaction turns in the *unbiased* simulator. Additionally, we apply the evaluation metrics mean squared error (MSE) and mean absolute error (MAE), both of which are widely used for the rating prediction task. Finally, Accuracy (ACC) and Click-ACC are also used to show the accuracy of the predicted ratings and the predicted click/skip behavior generated by the click model, which maps high ratings into *click*s and low ratings into *skip*s.

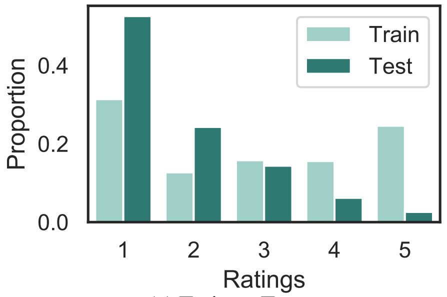
4.7 Experimental Results

4.7.1 Effect of Interaction Bias

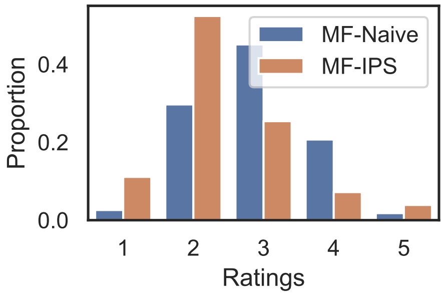
Recall that in Figure 4.3, we illustrate the propagation of the effect of bias from user interactions to reach the produced policies. We analyze the effect of bias on the task of predicting the rating matrix. In Table 4.1, we find that MF-IPS outperforms MF-Naive with metrics MSE, MAE and ACC on two real datasets[4] and synthetic data with $\alpha = 1$.

Moreover, for a better understanding of how bias affects MF, we analyze the complete user-item rating matrices generated by MF-IPS and MF-Naive. For the sake of brevity, we only present the analysis of positivity bias on the Yahoo!R3 and synthetic data. Figures 4.5a and 4.6a show the rating distributions of the MNAR logged data (Train) and the unbiased data (Test for MCAR data of Yahoo!R3 or GT for the synthetic data). Positivity bias in the logged data is adequately demonstrated resulting in a large "*oversampling*" of the higher ratings. Figures 4.5b and 4.6b show the rating distributions of the complete user-item rating matrices generated by MF-IPS and MF-Naive learned from the logged

[4]The results are similar to those reported in [117], but slightly different because we consider matching the real rating distributions.

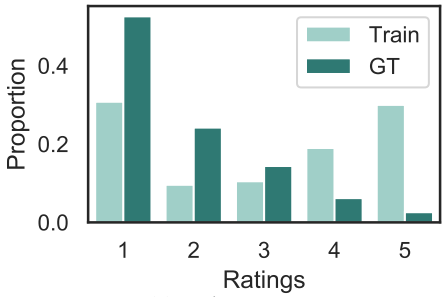


(a) Train vs Test

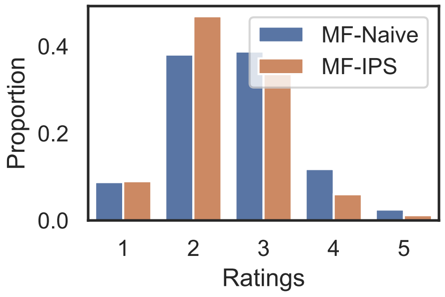


(b) MF-IPS vs. MF-Naive

Figure 4.5: Rating distributions on Yahoo!R3 dataset.



(a) Train vs Test



(b) MF-IPS vs. MF-Naive

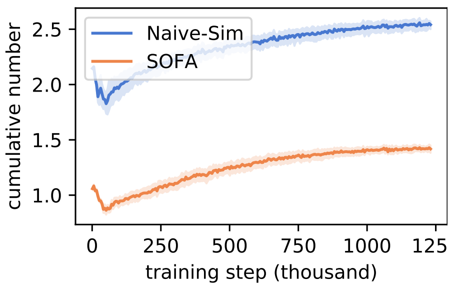
Figure 4.6: Rating distributions on the synthetic data with $\alpha = 1$.

data. We find that MF-Naive tends to overestimate ratings, and this in turn confirms our theoretical analysis in Section 4.3.2. MF-IPS can mitigate this kind of bias to some extent, shown here as a larger number of lower ratings than with MF-Naive. We notice a mismatch in the rating distributions between the true rating matrices and the generated rating matrices with ratings concentrating at 2 for MF-IPS or 3 for MF-Naive. The main reason is that MF models learned from the sparse logged data still suffer from large prediction errors, and predicting ratings as the median yields the minimal loss (*e.g.,* MSE loss).

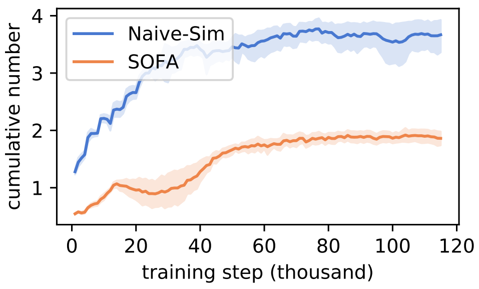
To conclude, interaction bias affects the prediction of the rating matrix based on logged data. Thus, any simulator using such a prediction model will also be biased, and the quality of policies trained using such a biased simulator will be affected.

4.7.2 Evaluation Results on Resulting Policies

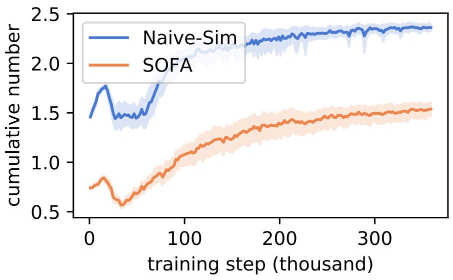
Two DQN policies equipped with the same networks first interact with two simulators, one with IBMS named SOFA and one without IBMS named Naive-Sim, and update their parameters from the interactions. Figure 4.7 shows the learning curves of these DQN policies, which track average cumulative number of clicks over 10-turn interactions with given simulators SOFA and Naive-Sim on Yahoo!R3, Coat and synthetic dataset with $\alpha = 1$.



(a) Yahoo!R3.



(b) Coat.



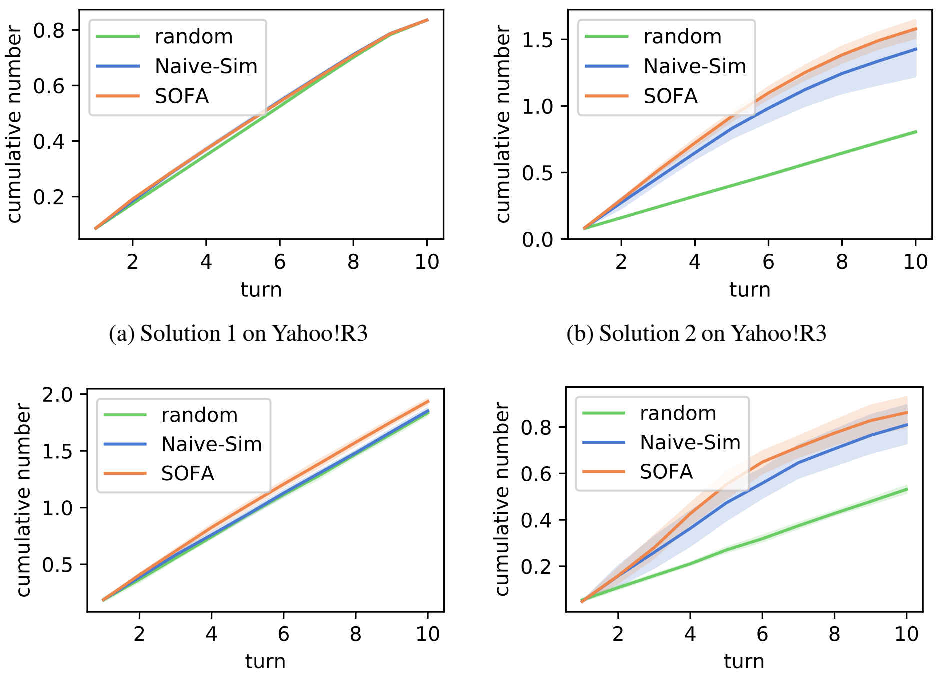
(c) Synthetic data with $\alpha = 1$.

Figure 4.7: Learning curves tracking average cumulative number of clicks over 10-turn interactions with given simulators SOFA and Naive-Sim. Results are an average of 10 independent runs, lines show mean performance, and shaded areas are confidence intervals.

We notice that learning curves show a downward trend at the begining of learning, because the simulators follow the basic hypothesis for RSs that users would dislike repeated recommended items and directly skip them. The duplicate recommendation is detrimental to novelty and we should avoid it [120]. We can observe that these policies converge after multiple learning steps, and the cumulative numbers of click for the policies resulting from using Naive-Sim are consistently higher than SOFA over the whole learning process. It is noteworthy that the learning curves are based on the number of clicks received during training; they are an unreliable estimate of actual performance due to bias.

Figure 4.8 shows the evaluation results on the Yahoo!R3 and Coat datasets with two solutions of evaluation on the sparse MCAR data: (1) Solution-1: **Limiting Action Selection**, (2) Solution-2: **Completing the Rating Matrix**.

DQN policies resulting from using simulators outperform the random recommendation policy on two real datasets. For Solution-1, the gap of the cumulative number of clicks over interactions between the different policies is not significant on Yahoo!R3. The most plausible reason is that the limited action candidate set is too distinct from the items that policies would actually recommend. For Solution-2, the produced DQN policies clearly outperform random recommendation policies. In the first turn of interaction, policies



(a) Solution 1 on Yahoo!R3

(b) Solution 2 on Yahoo!R3

(c) Solution 1 on Coat

(d) Solution 2 on Coat

Figure 4.8: Evaluation results of the produced policies on Yahoo and Coat.

show the same results because DQN-based policies randomly recommend an item in the first turn when the initial user state is empty. Then the policies recommend the item following the ϵ-greedy strategy to choose the action with highest Q-value, and obtain better performance than the random recommendation policy.

DQN policies resulting from using SOFA perform better than the policies resulting from using Naive-Sim in most cases, except for the evaluation results for Solution-1 on the Yahoo!R3 dataset, most likely because the limited action candidate set results in very similar recommendations for all the different policies. The evaluation results on the *debiased* simulator show a reversal of relative differences compared to the learning curves. This again supports our analysis on Section 4.3.2 that the simulator without IBMS overestimates ratings on average and simulates users to click more often because of bias. This reversal also answers RQ5.1 positively: Interaction bias in logged data does affect a simulator.

We also present evaluation results on the synthetic data with the observation of the logged data fully associated with positivity bias ($\alpha = 1$) by deploying the resulting policies in the unbiased simulator directly created with the complete true rating matrix GT, shown in Figure 4.9. We observe results consistent with those on the real-world datasets: DQN policies outperform the random recommendation policy, and the policies resulting from using SOFA outperform the policy resulting from using Naive-Sim. Therefore, we

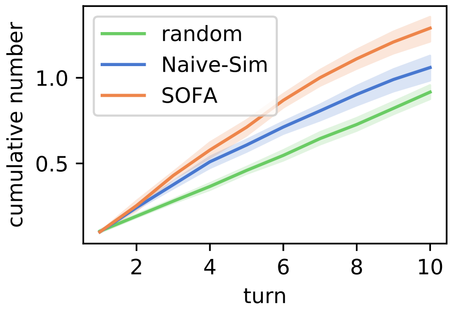


Figure 4.9: Policy evaluation with cumulative number of clicks on the synthetic data with $\alpha = 1$.

answer RQ5.2 positively: the proposed SOFA with IBMS does mitigate enough bias to result in better-performing policies.

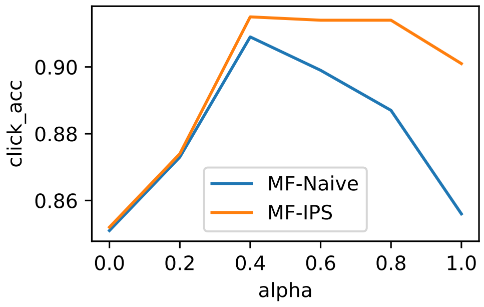
4.7.3 Effect of the Intensity of Bias

To answer RQ5.3, we evaluate the simulated feedback and the resulting policies in the synthetic simulator built with the complete true rating matrix GT.

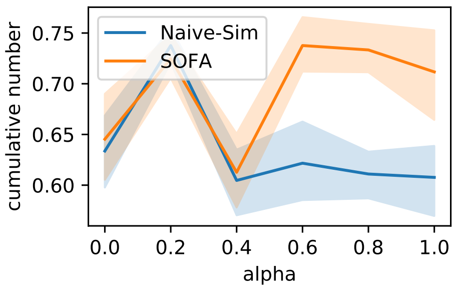
Figure 4.10a shows the performance of simulated feedback with evaluation metric Click-ACC. When α equals 0 with no bias in the logged data, the performance of simulated feedback generated from the rating matrices completed by MF-IPS and MF-Naive are similar. With the increase of the bias in logged data, MF-IPS consistently outperforms MF-Naive. Both achieve their best performance when $\alpha = 0.4$. With the increase of bias with α in range of 0.4–1.0, the performance of MF-Naive gradually decreases because the bias in logged data leads to grossly incorrect parameters estimation and rating prediction models. In contrast, MF-IPS with IBMS is more robust, which once again answers RQ5.2 positively.

Figures 4.10b and 4.10c show the cumulative numbers of clicks for policies over 5 and 10-turn interactions respectively. When α is bigger than 0.5, leading to a large degree of bias in the logged data, SOFA can lead to a better policy over 5 and 10-turn interactions. When α is smaller, SOFA performs worse over 10-turn interactions, but similar over 5-turn interactions. A plausible explanation is that IPS suffers from high variance and may underestimate ratings on average due to overweighting the lower logged ratings. This underestimation of the ratings results in less positive feedback than the real case, and further affects the policy to obtain similar performance over 5-turn interactions, but worse on turn-10.

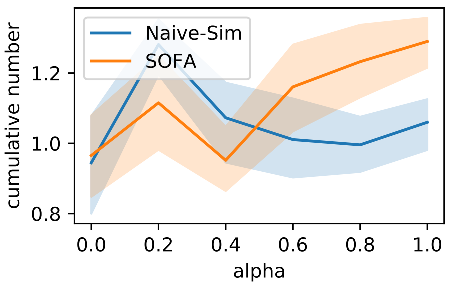
Finally, we answer RQ5.3: When the degree of bias in the logged data is very high, IBMS with using an IPS estimator can efficiently mitigate the bias from affecting the simulators and their produced policies. However, a minor flaw is that the IPS estimator used in IBMS can suffer from variance when there is very little bias in the logged data.



(a) Simulated feedback



(b) Clicks over 5 turns



(c) Clicks over 10 turns

Figure 4.10: Varing the intensity of bias. (a) shows the performance of the simulated feedback with Click-ACC. (b) and (c) report the evaluation results for the policies over 5- and 10-turn interactions.

4.8 Conclusion

In this paper, we have analyzed the phenomenon that interaction bias in logged data affects RL4Rec simulators and the policies they produce. To mitigate the effect of bias, we have proposed the *intermediate bias mitigation step* (IBMS), an intermediate step between the logged data and the learned prediction model. Furthermore, we have introduced a novel way of evaluating the effect of bias on the final policy performance of a simulator. Experimental results have revealed that (1) interaction bias in logged data affects a simulator, (2) the proposed IBMS can mitigate the bias, especially in the case of serious bias. We have combined IBMS and the newly proposed evaluation method, in a novel Simulator for OFfline leArning and evaluation (SOFA) to help researchers in the field develop and evaluate reinforcement learning for recommendation (RL4Rec) algorithms while mitigating the effects of interaction bias.

With these contributions, we can now answer the thesis-level research questions RQ5 and RQ6 positively: By applying the proposed debiasing step in the simulation pipeline, we can mitigate the effect of bias present in logged user interactions on the simulator for RL4Rec; furthermore, the proposed evaluation approach for simulators considers the performance of RL4Rec methods optimized with these simulators.

While we think that the IBMS is an important contribution to RL4Rec, SOFA only simulates the single-item recommendation scenario, where only one item is recommended at once. In practice, RSs often recommend multiple items at once, and thus future work could further consider the effect of interaction bias on simulators for multi-item recommendation scenarios.

5 State Encoders in Reinforcement Learning for Recommendation

Methods for reinforcement learning for recommendation (RL4Rec) are increasingly receiving attention as they can quickly adapt to user feedback. A typical RL4Rec framework consists of (1) a state encoder to encode the state that stores the users' historical interactions, and (2) an RL method to take actions and observe rewards. Prior work compared four state encoders in an environment where user feedback is simulated based on real-world logged user data. An attention-based state encoder was found to be the optimal choice as it reached the highest performance. However, this finding is limited to evaluation-simulators that do not debias logged user data. In this chapter, we address the thesis-level research question:

RQ7 Can the findings regarding the optimal choice of state encoders in RL4Rec methods generalize to the debiased simulation?

Besides, this finding is also limited to the actor-critic method and four state encoders. In response to all these shortcomings, we reproduce and expand on the existing comparison of attention-based state encoders (1) in the publicly available debiased RL4Rec SOFA simulator with (2) a different RL method, (3) more state encoders, and (4) a different dataset. Importantly, our experimental results indicate that existing findings do *not* generalize to the debiased SOFA simulator generated from a different dataset and a Deep Q-Network (DQN)-based method when compared with more state encoders.

5.1 Introduction

With the development of interactive recommender systems (RSs), RL4Rec is receiving increased attention as reinforcement learning (RL) methods can quickly adapt to user feedback [4, 77]. RL4Rec has been applied in a variety of domains, such as movie [154, 159], news [167], and music recommendations [99]. A typical process flow of RL4Rec starts

This chapter was published as: J. Huang, H. Oosterhuis, B. Cetinkaya, T. Rood, and M. de Rijke. State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2738–2748. ACM, July 2022.

with an action of the system, which is an item being recommended to the user. Subsequently, user interaction with the item is returned as feedback (*e.g.,* dwell time, a rating, or a click) to the system, which then interprets the feedback as a reward signal. Finally, with this new interaction, the system updates a state representation that keeps track of the user's historical interactions with the recommended items. The cycle then repeats as the system again tries to recommend the best item to the user based on its updated state representation. The goal of RL4Rec is to optimize the system so as to achieve the maximum cumulative reward.

An RL4Rec framework typically consists of two parts: (1) the *state encoder* that encodes the state – a user's historical interactions – into a dense representation that is used to estimate the user's preference and the value of state-action pairs; and (2) an *RL method* (*e.g.,* the DQN [94] or the actor-critic [76] method) that is applied to generate actions based on an estimated state-action value function and observed reward. While RL4Rec methods have achieved good performance, the effect of the state encoder on RL4Rec methods has rarely been explicitly looked at. To bridge this gap, Liu et al. [83] compared four state encoders in a simulated RL4Rec environment and concluded that an attention-based state encoder leads to the best recommendation performance. Their findings revealed that the choice of state encoders is important for effective RL4Rec and, accordingly, this shows that research into state encoders could further improve the performance of RL4Rec methods. However, the analysis of Liu et al. [83] is limited to the actor-critic method and only four different state encoders. Moreover, their evaluation was based on simulated user feedback that was directly inferred from logged user data, which is typically subject to heavy selection bias, *e.g.,* popularity bias [123]. Consequently, due to a lack of any bias correction, it is very likely that the results and findings of Liu et al. [83] are also affected by the selection bias present in the data.

In response to these shortcomings, we reproduce the work by Liu et al. [83] and generalize its findings concerning state encoders in the following directions:

(1) *Different simulated environments*: The simulated user feedback used in [83] is generated from logged user data which is inevitably subject to user selection bias, *e.g.,* popularity bias [104]. Chapter 4 pointed out that simulators that do not debias logged user data yield RL4Rec methods that are heavily affected by selection bias [52]. Hence, we use our proposed SOFA [52] – the only publicly available debiased simulator [12] – to mitigate the effect of selection bias on the resulting RL4Rec methods.

(2) *Different RL method*: DQN is the most popular RL method used in RL4Rec [24, 26, 58, 81, 82, 85, 105, 130, 154, 164, 166, 167]; it is structurally simpler than the actor-critic method by only optimizing one objective. Thus, it matters to find out whether comparisons of state encoders generalize to DQN-based RL4Rec methods.

(3) *More state encoders*: Several typical neural networks – multi-layer perceptrons (MLPs), gated recurrent units (GRUs), and convolutional neural networks (CNNs) – are not considered as state encoders by Liu et al. [83]. We expand their comparison by adding these three state encoders based on widely used typical neural networks.

(4) *Different dataset*: Besides the Yahoo!R3 dataset [89] used by Liu et al. [83], we also use the Coat shopping dataset [117] to build the debiased SOFA simulator.

We report on our efforts to reproduce the main finding in [83]:

> *The attention state encoder for RL4Rec provides significantly higher performance than the bag of items (BOI), pairwise local dependency between items (PLD) and average (Avg) state encoders.*

Moreover, we investigate whether this finding generalizes in the four directions described above. Our experimental results show that Liu et al.'s finding *is* reproducible when applying a DQN method and evaluating in the debiased SOFA simulator on the Yahoo!R3 dataset. However, we also find that it does *not* generalize to debiased simulations generated from the Coat shopping dataset [117].

Our study addresses the following chapter-level research questions that are intricately linked to the thesis-level research question RQ7:

RQ7.1 Does Liu et al.'s main finding generalize to the DQN-based RL4Rec methods when evaluating in the debiased SOFA simulator and compared with more state encoders, *i.e.,* with the MLP, GRU and CNN state encoders?

RQ7.2 Does Liu et al.'s main finding generalize to a debiased simulation based on a different dataset?

RQ7.3 Should the choice of activation function be taken into account when using the MLP-based state encoder for RL4Recs?

5.2 Related Work

RL4Rec methods. Deep RL methods (*e.g.,* DQN, actor-critic, and REINFORCE) are able to handle high-dimensional spaces and are therefore particularly suitable for RSs with large state spaces where the user state involves combinatorial user interaction behavior [4]. DQN has been the most popular choice among the RL4Rec methods [24, 26, 58, 81, 82, 85, 105, 130, 154, 164, 166, 167]. Chen et al. [24] integrate stratified sampling action replay and approximated regretted rewards with Double DQN to stabilize the RL4Rec methods in dynamic environments. Zhao et al. [164] incorporate positive and negative feedback in a RL4Rec method. Chen et al. [26] propose a cascading DQN method to obtain a combinatorial recommendation policy with large item space. Liu et al. [82] introduce a supervised signal to enable stable training of RL4Rec methods. Others use DQN in special recommendation scenarios, *e.g.,* for news [167], movies [154], education [85], projects [105], slates [58, 130], or mobile users [81]. REINFORCE and actor-critic are the other two important methods adapted in RL4Rec. REINFORCE is a policy gradient method that directly updates the policy weights [141]. Liang [74] adapts REINFORCE to find a path between users and items in an external heterogeneous information network. REINFORCE with importance sampling can be used to correct for biases caused by only observing feedback on items recommended by other RSs [22, 86]. Additionally, REINFORCE is commonly used in conversational RSs [39, 129] and explainable RSs [145]. Actor-critic combines REINFORCE and the value-based method [76], thus benefiting from both components; it is able to handle large action spaces in RSs [33]. Actor-critic has been used for diverse recommendation tasks [162, 163] and domains [153, 159].

In general, DQN is the most popular RL method used in RL4Recs and has a simpler structure than actor-critic. Accordingly, we use DQN and investigate whether the findings on actor-critic based RL4Rec in [83] generalize to DQN based RL4Rec.

State encoders. Neural networks are widely used in collaborative filtering (CF) based recommendation methods; popular choices are multi-layer perceptrons (MLPs) [27, 44], convolutional neural networks (CNNs) [45], recurrent neural networks (RNNs) [48, 142, 152] and attention [25, 50]. Based on logged user behavior, these methods usually use a neural network to generate a dense vector that captures user preferences and can be further used to infer the users' preferences over items. That makes it suitable to adapt these recommendation methods in the state encoders. Most of the above RL4Rec methods use neural networks, such as variants of RNNs [22, 164], to construct the state encoder and generate state representation, which can subsequently be used by the previously discussed RL methods. However, the effect of state encoders has rarely been explored explicitly. To the best of our knowledge, Liu et al. [83] are the first to compare the effects of different state encoders in RL4Rec methods. We continue this research direction by reproducing and generalizing Liu et al.'s comparison.

Debiasing recommendations. Bias is prevalent in interactions with RSs, such as users choosing to rate certain items more often (self selection bias) [104, 122] and RSs showing certain items to users more often (algorithmic selection bias) [41]. As a result, user preference prediction may be biased and over-specialization [3], consequently, filter bubbles [95, 102] and unfairness [21] may occur. To correct for bias, debiasing methods may be applied, such as the error-imputation-based method [122], inverse propensity scoring (IPS) [49], and the doubly robust method [63, 110]. IPS is the most popular method and widely used in debiasing recommendations [22, 23, 62, 86, 117]. Corrections of the debiased methods may lead to substantially improved prediction performance [117].

RL4Rec simulators. The usage of simulated RL4Rec environments is widespread [12, 57, 71, 111, 118, 119, 156, 165, 167] and for a good reason: RL4Rec methods learn by directly interacting with users but the online nature of this learning process brings risks and limitations: (1) in practice, the user experience can be negatively affected during the early stages of the learning process; and (2) research and experimentation with RL4Rec systems is often infeasible since most researchers have no access to real interactions with live users. RS simulators mitigate these issues as they allow RS developers and researchers to optimize and evaluate their RL4Rec methods on simulated user behavior [12, 57, 111, 118]. Some simulators generate user behavior based on fully synthetic data (*e.g.,* generated from a Bernoulli distribution [111]). These have been critiqued for oversimplifying user behavior [12, 118]. Alternatively, to match real user behavior more closely, other simulators generate user behavior based on logged user data [57, 118, 165]. While these simulators are widely accessible, most ignore the interaction biases present in the logged user data from which they generate simulated user behavior. In Chapter 4, we have pointed out that simulators that do not debias logged user data result in RL4Rec models that are also heavily affected by the selection biases [52]. We argue that, as a result, findings based on the outcomes of such biased simulators can be misleading because the effect of the interaction biases extend to the results underlying such findings. To mitigate the effect of bias, the SOFA environment [52] introduced in Chapter 4 applies inverse propensity scoring (IPS) to reduce selection bias in logged user data when learning user preference and thus provides a debiased simulator. To the best of our knowledge, SOFA is the only publicly available debiased simulator. Therefore, we use SOFA to train and evaluate RL4Rec methods with different state encoders.

5.3 Preliminaries – RL4Rec

RL4Rec methods commonly model the recommendation task as a Markov decision process (MDP), where optimization is based on interactions between the RS (*i.e.,* the agent) and users (*i.e.,* the environment). The elements of an MDP for RL4Rec are:

State space $\mathcal{S}$: A state s_t^u stores the interaction history of user u at t-th turn. For clarity and brevity, we omit the superscript u when the user is clear from the context. The state s_t consists of the items recommended by the RS and the corresponding user feedback (*e.g.,* click or skip), denoted as $s_t = ([i_1,i_2,...,i_t], [f_1, f_2, \dots, f_t])$. In turn $t+1$, the RS takes an action based on the information represented in state s_t. The state s_0^u is always initialized as empty, denoted as $s_0^u = ([\,],[\,])$.

Action space $\mathcal{A}$: The action a_t is to recommend an item i_t to user u by the RS based on state s_{t-1} in turn t. Similar to the setup of Liu et al. [83], in the SOFA simulator the RS only recommends one item to the user at every turn.

Reward $\mathcal{R}$: The immediate reward $r(s_{t-1},a_t)$ is generated according to user's feedback f_t (*e.g.,* skip or click) on a_t.

Transition probability $\mathcal{P}$: In turn $t+1$, SOFA receives an item i_{t+1} being recommended from the RS and assumes that the state s_t transitions deterministically to the next state s_{t+1} by appending item i_{t+1} and the corresponding user feedback f_{t+1}, denoted as $s_{t+1} = ([i_1,i_2,...,i_{t+1}],[f_1,f_2,...,f_{t+1}])$.

Discount factor γ: $\gamma \in [0,1]$ determines the degree to which the RS cares about future rewards: if $\gamma = 0$, the RS only takes the immediate reward into account when taking an action; if $\gamma = 1$, the sum of all future rewards is considered.

Generally, the RL4Rec method includes two components as shown in Figure 5.1: (1) the state encoder is applied to encode a state s into a dense representation that captures the user preference and is subsequently used to approximate the state-action value function $\widehat{Q}(s,a;\theta)$; for every action $a \in \mathcal{A}$, $\widehat{Q}(s,a;\theta)$ represents the expected reward following the recommendation of item a in state s; and (2) the RL method decides which action to take based on the state representation, and chooses how the parameters of the policy and state encoder models should be updated according to the rewards received from the user.

While the RL method chooses items to recommend to the user, it bases its decisions on the state representations provided by the state encoder. Therefore, the performance of an RL4Rec system heavily relies on the functioning of the state encoder. As a result, understanding how the choice of state encoder should be made is central to RL4Rec.

5.4 Original State Encoder Comparison

Liu et al. [83] follow the RL4Rec framework detailed in Section 5.3 and apply an actor-

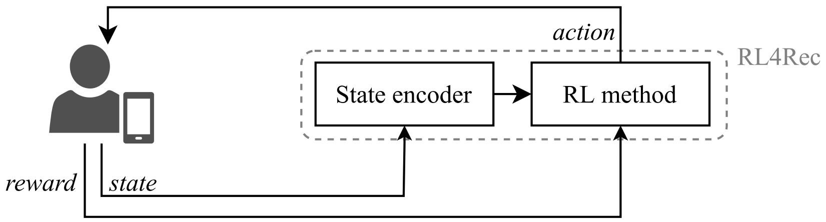


Figure 5.1: The general framework of RL4Rec.

critic RL method to take actions and update the model parameters. The applied actor-critic method comprises two components: (1) the actor network follows the policy $\pi_{\theta^A}(s_{t-1}) \in \mathbb{R}^d$ and takes the action a_t, *i.e.,* to recommend item i with the maximum ranking score $\boldsymbol{q}_i^\top \pi_{\theta^A}(s_{t-1})$, where $\boldsymbol{q}_i$ denotes the embedding of item i; (2) the critic network estimates state-action value function $\widehat{Q}(s,a;\theta^C)$ as the approximation of the true state-action value function that represents the merits of the recommendation policy generated by the actor network. The target network technique is also adopted, where an identical actor network with policy $\pi_{\theta^{A'}}$ and an identical critic network with state-action value function $\widehat{Q}(s,a;\ \theta^{C'})$ are used. The recommendation agent makes use of experience replay and employs a replay memory $\mathcal{D}$ to store the agent's experience, *i.e.,* the user interactions with the recommended items in the RL4Rec domain. Given transitions $(s_{t-1},a_t,r_t,s_t) \in \mathcal{D}$ generated based on the interactions between the user and recommendation policy π_{θ^A}, *i.e., $a_t \sim \pi_{\theta^A}(s_{t-1})$*, the parameters θ^A of the actor network and θ^C of the critic network are updated as:

$$\begin{aligned} \theta^A &\leftarrow \theta^A + \alpha^A \widehat{Q}(s_{t-1},a_t;\theta^C)\nabla_{\theta^A}\log\pi_{\theta^A}(s_{t-1}), \\ \theta^C &\leftarrow \theta^C + \big(\alpha^C(r_t + \gamma\widehat{Q}(s_t,a_{t+1};\theta^{C'}) \\ &\qquad\qquad - \widehat{Q}(s_{t-1},a_t;\theta^C))\nabla_{\theta^C}\widehat{Q}(s_{t-1},a_t;\theta^C)\big), \end{aligned} \tag{5.1}$$

where α^A and α^C denote the learning rates for the actor network and the critic network, respectively, and $a_{t+1} \sim \pi_{\theta^{A'}}(s_t)$. The target network is updated following the soft replace technique: given a soft-replace parameter τ, the parameters $\theta^{A'}$ of the actor network and $\theta^{C'}$ of the critic network are updated as follows:

$$\theta^{A'} \leftarrow \tau\theta^A + (1-\tau)\theta^{A'}, \qquad \theta^{C'} \leftarrow \tau\theta^C + (1-\tau)\theta^{C'}. \tag{5.2}$$

Liu et al. [83] consider two types of state encoder methods for representing states and approximating the state-action value functions, with and without user embedding $\boldsymbol{p}_u$. First, they introduce an item-to-item collaborative filtering method, DRR-p, without taking user embeddings into account, which uses an element-wise product to capture the pairwise local dependency between items:

$$\boldsymbol{s}_t = [\boldsymbol{q}_{i_1},\boldsymbol{q}_{i_2},\ldots,\boldsymbol{q}_{i_t},\{w_i\boldsymbol{q}_i \otimes w_j\boldsymbol{q}_j \mid i,j \in \{i_1,i_2,\ldots,i_t\}\}], \tag{5.3}$$

where $\otimes$ denotes the element-wise product; and the scalars w_i and w_j indicate the importance weights of items i and j, respectively. Additionally, three state encoders, DRR-u, DRR-ave and DRR-att, with user embeddings are introduced and outperform the state encoder DRR-p without user embeddings: (1) the element-wise product on user-item embedding pairs is incorporated: $\boldsymbol{s}_t = [\{\boldsymbol{p}_u \otimes w_i \boldsymbol{q}_i \mid i \in \{i_1, i_2, ..., i_t\}\}]$; and (2) to reduce computational costs, the weighted average pooling schema is used to aggregate the item embeddings: $\boldsymbol{s}_t = [\boldsymbol{p}_u, \boldsymbol{p}_u \otimes \{\text{ave}(w_i \boldsymbol{q}_i) \mid i \in \{i_1, i_2, ..., i_t\}\}]$, where ave$(\cdot)$ denotes the average pooling operator. Finally, (3) an attention network is applied:

$$\boldsymbol{s}_t = [\boldsymbol{p}_u, \boldsymbol{p}_u \otimes \{\text{ave}(a_{u,i} \boldsymbol{q}_i) \mid i \in \{i_1, i_2, ..., i_t\}\}], \tag{5.4}$$

$$a_{u,i} = \frac{\exp(a'_{u,i})}{\sum_{i' \in \{i_1, i_2, ..., i_t\}} \exp(a'_{u,i'})}, \tag{5.5}$$

$$a'_{u,i} = \text{ReLU}(([\boldsymbol{p}_u, \boldsymbol{q}_i] \boldsymbol{W}_2) + b_2) \boldsymbol{W}_1 + b_1, \tag{5.6}$$

where the weight matrices $\boldsymbol{W}_1, \boldsymbol{W}_2$ and the bias vectors b_1, b_2 project the input into a hidden layer; ReLU is the activation function for the hidden layer.

Given the state representation $\boldsymbol{s}_t$, the ranking score of item i, *i.e.*, $\boldsymbol{p}_i^\top \pi_{\theta^A}(s_t)$, can be used to execute policy $\pi_{\theta^A}(s_t)$ and approximate state-action value function $\widehat{Q}(s_t, a; \theta^C)$. Consequently, the resulting actor-critic-based RL4Rec method can interact with the (simulated) users and update the parameters iteratively. Liu et al. [83] compare the actor-critic based RL4Rec method with four state encoders, DRR-p, DRR-u, DRR-ave and DRR-att, in simulators generated from two datasets [43] containing temporal information and two datasets not containing temporal information [38, 89]. They conclude that: (1) state encoders that utilize user embeddings outperform state encoders without user embeddings; (2) the average pooling schema can decrease the dimensionality of the state representation to reduce overfitting and improve recommendation performance; and (3) the attention-based state encoder provides the best performance among the four state encoders introduced above.

5.5 Our Reproduced State Encoder Comparison

Having specified the setting of Liu et al. [83]'s study (See Section 5.4), we generalize Liu et al.'s finding to four directions and can summarize the following key differences:

(1) **Different simulated environments**: We adopt SOFA, the debiased simulator, which mitigates the effect of bias present in logged data when generating user preferences on items; in contrast, Liu et al. [83] use a simulation directly generated from logged data without considering bias.

(2) **Different RL method**: We apply DQN, which is widely used in RL4Rec research and has a simpler structure with optimizing only one objective, whereas Liu et al. [83] apply the actor-critic method.

(3) **More state encoders**: Besides the four state encoders proposed by Liu et al. [83], we expand the comparison by adding three more state encoders based on typical neural network architectures: MLP, GRU and CNN, which are widely used in recommendation methods to generate representations according to historical user interactions.

(4) **Different dataset**: Our comparison uses the Yahoo!R3, which is also used by Liu et al. [83], as well as the Coat shopping dataset [117], which is not considered in [83]. To the best of our knowledge, these two datasets are the only publicly available datasets that can be used to unbiasedly simulate recommendations, since part of their data is gathered on randomized recommendations. Unfortunately, there are two more datasets used by Liu et al. [83] that cannot be used in SOFA due to a lack of randomized data for debiasing.

It is crucial to understand whether the choice of state encoder is important, and if so, what factors should be considered when making this choice. In particular, the aim of our reproducibility study is to analyze whether the choice of state encoder is robust w.r.t. the effect of bias, the choice of RL method, and the sources of data used. The differences listed above allow us to address this aim and investigate whether the findings of Liu et al. [83] generalize along these dimensions.

Below, we describe the setting in which we reproduce and expand on the comparisons performed by Liu et al. [83]. Section 5.5.1 details the debiased SOFA simulator that we use, Section 5.5.2 explains the DQN RL method that is applied, and finally, Section 5.5.3 lists the state encoders included in our comparison.

5.5.1 Simulator for Offline Learning and Evaluation (SOFA)

To mitigate the effect of bias present in logged data, in Chapter 4, we propose a debiased simulator, named SOFA, which consists of two components [52]: (1) a debiased user-item rating matrix to present user preferences for items, and (2) a user choice model to simulate user feedback and generate the next state and the immediate reward. The bias mitigation step is applied between the logged data and the learned user preference prediction model, thereby mitigating the bias originating from the logged data from affecting the user preference prediction model. User behavior (*e.g.,* ratings) could be affected by various forms of selection bias, *e.g.,* users tend to rate more popular items (*popularity bias*) [104, 123] or the items that they expect to enjoy beforehand (*positivity bias*) [104]. This is generally modelled by decomposing the probability of observing a rating $y_{u,i}$ given by user u on item i into (1) the preference $P(y_{u,i})$, *i.e.,* the distribution over rating values the user u would give to item i; and (2) the propensity $P(o_{u,i})$, *i.e.,* the probability of observing any rating from user u for item i in the dataset. The assumed model is thus:

$$P(o_{u,i}, y_{u,i}) = P(o_{u,i}) P(y_{u,i}), \tag{5.7}$$

where $o_{u,i}$ denotes the observation indicator: $o_{u,i} = 1$ if the rating $y_{u,i}$ is observed, otherwise, $o_{u,i} = 0$ indicates a rating is missing. Due to bias, certain ratings are more likely to be observed than others. In other words, $P(o_{u,i})$ is not uniform over all user-item pairs. As a result, naively ignoring the propensities during evaluation or optimization gives more weight to the user-item pairs that are overrepresented due to bias [117], *e.g.,* giving the most weight to the most popular items. In turn, this results in biased user rating predictions $\hat{y}_{u,i}$ that fail to match the true user ratings $y_{u,i}$. The bias mitigation step of SOFA applies inverse propensity scoring (IPS) [59] to inversely weight ratings according to the corresponding observation probabilities so that, in expectation, each user-item pair is represented equally. Let $\delta(\hat{y}_{u,i}, y_{u,i})$ indicate the loss resulting from the match between the predicted rating $\hat{y}_{u,i}$ and true rating $y_{u,i}$ [117]:

$$\mathbb{E}[\mathcal{L}_{\text{IPS}}] \propto \mathbb{E}\left[\frac{\delta(\hat{y}_{u,i}, y_{u,i})}{P(o_{u,i}=1)}\right] = \frac{\mathbb{E}[o_{u,i}]\delta(\hat{y}_{u,i}, y_{u,i})}{P(o_{u,i}=1)} = \delta(\hat{y}_{u,i}, y_{u,i}). \tag{5.8}$$

Therefore, using the IPS debiasing method, SOFA can learn debiased user preferences for items and mitigate the effect of bias on the resulting simulated user behavior and the final produced RL4Rec methods [52].

Before the interaction starts, SOFA uniformly randomly samples a batch of users and initializes their states as empty; then SOFA interacts with RL4Rec methods over ten turns. Within SOFA, the RL4Rec methods aim to maximize the cumulative number of clicks received over ten interaction turns, and are accordingly evaluated on the cumulative number of clicks they receive over ten interaction turns. Furthermore, SOFA provides a general DQN-based RL4Rec framework, which Section 5.5.2 describes in detail.

5.5.2 Deep Q-Network based Recommendation

Deep Q-Networks (DQNs) [94] are based on Q-learning, one typical value-based RL method [131], while the actor-critic methods integrate a value-based method with the policy gradient REINFORCE method [141]. As a result, DQNs have a simpler structure than actor-critic methods by only optimizing one objective; thus, while actor-critic methods are potentially more powerful for handling large state and action spaces, DQNs can be more data-efficient. DQNs have been widely used in RL4Rec to improve recommendation performance [24, 26, 58, 81, 82, 85, 105, 130, 154, 164, 166, 167]. For these reasons, we follow SOFA and choose to use the basic DQN for our reproducibility study. We optimize the DQN by fitting its predicted state-action function $\widehat{Q}(s,a;\theta)$ to the expected discounted cumulative reward $\sum_t \gamma^t r_t$. To stabilize the training process, DQN introduces a behavior network separate from the target network. Here, we apply a state encoder as the behavior network and an identical state encoder as the target network. These two state encoders have the same structure and use the same item embeddings, but are updated in different ways. Moreover, DQN makes use of experience replay and employs a replay memory $\mathcal{D}$ to store the agent's experience, *i.e.,* the user interactions with the recommended items in the RL4Rec domain.

Given a transition $(s_{t-1}, a_t, r_t, s_t) \in \mathcal{D}$, the behavior network estimates Q-value function $\widehat{Q}(s_{t-1}, a_t; \theta)$ on the given state-action pair (s_{t-1}, a_t), where θ denotes the parameters of the behavior network; the target network is used to estimate Q-value function $\widehat{Q}'(s_t, a; \theta')$ for any action $a \in \mathcal{A}$ given state s_t, with the parameters θ' fixed and periodically copied from θ in the behavior network. Following Mnih et al. [94], the parameters θ of the behavior network are updated by minimizing the following smooth L1 loss function for steady gradients with the Adam optimizer:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_{t-1}, a_t, r_t, s_t) \sim D} \begin{cases} 0.5(\delta^{\text{TD}})^2 & \text{if } |\delta^{\text{TD}}| < 1, \\ |\delta^{\text{TD}}| & \text{otherwise.} \end{cases} \tag{5.9}$$

$$\delta^{\text{TD}} = r_t + \gamma \max_a \widehat{Q}'(s_t, a; \theta') - \widehat{Q}(s_{t-1}, a_t; \theta). \tag{5.10}$$

Note that the parameters θ' of the target network are not updated in each learning step, but are periodically replaced by θ after multiple learning steps.

5.5.3 State Encoders in Our Comparison

As described in Section 5.3, the state encoder is used to generate representations of the state that can be used as input to the approximated state-action value function. The choice of state encoder can have a large impact on the performance of the RL4Rec system [83]. Accordingly, it is crucial to select an appropriate and effective state encoder. Since Liu et al. [83] have not made their source code publicly available, we have reimplemented the four state encoders of their original comparison (see Section 5.4): DRR-p, DRR-u, DRR-ave and DRR-att. Due to the increasing importance of privacy and the fact that SOFA does not provide user information, we drop the user embedding w.r.t. the user id and add user feedback to the recommended items to obtain user preferences in these four state encoders, renamed as pairwise local dependency between items (PLD), bag of items (BOI), average (Avg) and Attention. Additionally, we consider three more typical neural networks – MLP, GRU and CNN – when constructing the state encoders.

We use $\boldsymbol{q}_i$ to denote the embedding of item i and $\boldsymbol{f}_i$ for the embedding of feedback $f_i \in \{0,1\}$ from the user on the item i. Given state $s_t = ([i_1,i_2,\dots,i_t],[f_1,f_2,\dots,f_t])$, we have the corresponding item embeddings $[\boldsymbol{q}_{i_1},\boldsymbol{q}_{i_2},\dots,\boldsymbol{q}_{i_t}]$ and feedback embeddings $[\boldsymbol{f}_{i_1},\boldsymbol{f}_{i_2},\dots,\boldsymbol{f}_{i_t}]$. The state-action value function $\widehat{Q}(s_t,a)$ can be approximated by the following state encoders:

BOI: Corresponding to DRR-u from Liu et al. [83], the state representation $\boldsymbol{s}_t^{\text{BOI}}$ is formulated as a list of weighted element-wise products of historical item embeddings and the corresponding feedback embeddings. Then, one linear layer is applied and the dimensionality of the output space is set to the number of items:

$$\begin{aligned} \boldsymbol{s}_t^{\text{BOI}} &= [\{w_i \boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1,i_2,\dots i_t\}\}], \\ \widehat{Q}(s_t,a) &= \boldsymbol{W}^\top \boldsymbol{s}_t^{\text{BOI}} + b. \end{aligned} \tag{5.11}$$

PLD: Corresponding to DRR-p from Liu et al. [83], the pairwise local dependency between items $e_{i,j}$ is also considered in modeling state representation $\boldsymbol{s}_t^{\text{PLD}}$:

$$\begin{aligned} \boldsymbol{s}_t^{\text{PLD}} &= [\{w_i \boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1,i_2,\dots i_t\}\}, \\ &\qquad \{e_{i,j} \mid i,j \in \{i_1,i_2,\dots,i_t\}\}], \\ e_{i,j} &= w_i(\boldsymbol{q}_i \otimes \boldsymbol{f}_i)^\top (\boldsymbol{q}_j \otimes \boldsymbol{f}_j) w_j, \\ \widehat{Q}(s_t,a) &= \boldsymbol{W}^\top \boldsymbol{s}_t^{\text{PLD}} + b. \end{aligned} \tag{5.12}$$

Avg: Corresponding to DRR-ave from Liu et al. [83], one linear layer is applied with no activation function and the dimensionality of the output space is set to the number of items:

$$\widehat{Q}(s_t,a) = \boldsymbol{W}^\top \text{ave}(\{\boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1,i_2,\dots,i_t\}\}) + b, \tag{5.13}$$

where ave$(\cdot)$ denotes the component-wise average operator on a set of vectors; and $\boldsymbol{W}$ and b are the weight and bias term of the linear layer, respectively.

MLP: Novel in our comparison, on top of Avg, we lift the linear assumption of state and state-action value function by applying a non-linear activation function σ, *e.g.,* tanh, ReLU, or sigmoid:

$$\widehat{Q}(s_t,a) = \sigma(\boldsymbol{W}^\top \text{ave}(\{\boldsymbol{q}_i \otimes \boldsymbol{f}_i \mid i \in \{i_1,i_2,\dots,i_t\}\}) + b). \tag{5.14}$$

CNN: Novel in our comparison, a basic CNN with one convolution layer and one max-pooling layer is applied; to compute $\widehat{Q}(s_t,a)$, a fully-connected layer is also adopted with the dimensionality being the number of items:

$$\widehat{Q}(s_t,a) = \boldsymbol{W}^\top \max(W_C(([\boldsymbol{q}_{i_1},\ldots,\boldsymbol{q}_{i_t},\boldsymbol{f}_{i_1},\ldots,\boldsymbol{f}_{i_t}]^\top))) + b, \tag{5.15}$$

where $\max(\cdot)$ denotes the max operator of the max-pooling layer; W_C indicates the weight function of a l-dilated convolution filter of size 3×3 and the activation function ReLU; and $\boldsymbol{W}$ and b are the weight and bias term of the fully-connected layer, respectively.

GRU: Novel in our comparison, a basic GRU layer and a dense layer are applied:

$$\begin{aligned} \boldsymbol{h}_k &= W_G(\boldsymbol{h}_{k-1},\boldsymbol{q}_{i_k}\otimes\boldsymbol{f}_{i_k}), \quad \forall k=1,2,\ldots,t \\ \widehat{Q}(s_t,a) &= \boldsymbol{W}^\top \boldsymbol{h}_t + b, \end{aligned} \tag{5.16}$$

where W_G indicates the weight function of the GRU unit with the activation funtion tanh; and $\boldsymbol{h}_0$ is set as a zero-vector. The hidden state vector $\boldsymbol{h}_k$ is computed conditioned on the previous hidden state vector $\boldsymbol{h}_{k-1}$ and the input $\boldsymbol{q}_{i_k}\otimes\boldsymbol{f}_{i_k}$.

Attention: Corresponding to DRR-att from Liu et al. [83], following [7] we insert an attention layer into the GRU-based state encoder:

$$a_k = \frac{\exp(a'_k)}{\sum_{k'=1}^{t}\exp(a'_{k'})}, \quad a'_k = (\boldsymbol{W}_A^\top \boldsymbol{h}_t)^\top \boldsymbol{h}_k, \tag{5.17}$$

$$\widehat{Q}(s_t,a) = \boldsymbol{W}^\top \left[\left(\sum_{k=1}^{t} a_k \boldsymbol{h}_k \right), h_t \right] + b, \tag{5.18}$$

where W_A denotes the weight function of the attention layer; a_k denotes the attention weight on the hidden state vector $\boldsymbol{h_t}$; and the attentive combination of all the hidden state vectors is used to compute the state-action value function $\widehat{Q}(s_t,a)$.

5.6 Experimental Setup

In this section, we describe the experiments performed to answer the research questions presented in Section 5.1.

Datasets and simulators. We use SOFA to generate two debiased simulations that simulate user behavior based on two real-world datasets: Yahoo!R3 [89] and Coat shopping [117], which – to the best of our knowledge – are the only publicly available datasets that include a uniformly randomly sampled test set that allows for unbiased evaluation. The number of users in the Yahoo!R3 and Coat shopping datasets are 15,400 and 290, respectively; and the number of items are 1,000 and 300, respectively. Both datasets include a biased training set and an unbiased test set: the training set contains ratings observed from *natural* real-world user behavior, whereas the test set contains ratings asked from users on uniformly randomly sampled items. Consequently, the training set is affected by the forms of bias present in standard user interactions, but the test set is unaffected by any selection bias since it relies on uniform random sampling. The

Table 5.1: List of hyperparameters for DQN and their values.

Hyperparameter	**Definition**	**Value**
Memory Size	The number of transitions stored in the replay memory.	6,000
Discount factor	Discount factor γ used in the DQN.	0.9
Epsilon	The minimal probability of recommending an item randomly when taking an action.	0.1
Epsilon decay frequency	The number of step with which the epsilon ϵ (initial value as 0.8) minus 0.1.	20,000
Minibatch size	The number of training cases randomly selected from replay memory and being used to update the parameters of policy.	128
Targetnet replacement frequency	The number of step with which the target network is updated.	20

simulations used for training RL4Rec methods are based on debiased user preferences generated from IPS-based rating prediction methods (Eq. 5.8) on the biased training set; in contrast, the evaluation of the RL4Rec methods is performed on the unbiased simulations generated from the unbiased test sets.

Hyperparameters. The required hyperparameters come in two kinds: (1) Hyperparameters of the used DQN: we follow the hyperparameters reported by Huang et al. [52] (see Table 5.1) and fix the values for the DQN based RL4Rec methods with different state encoders. (2) Hyperparameters used in the state encoders: the common hyperparameters are tuned per state encoder in the following ranges: learning rate $\eta \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ and the dimension of item embedding $d \in \{16, 32, 64\}$. Additionally, the dimensions of the weight functions in the CNN, GRU and attention state encoders are taken from $d' \in \{16, 32, 64\}$.

Evaluation metrics. As introduced in Section 5.5.1 we use the cumulative number of clicks received over 10 interaction turns in the unbiased simulated online environments to evaluate the performance of the state encoders in the Deep Q-Network based recommendation (DQN4Rec) method. The cumulative or average number of clicks is a common choice of metric [83, 164] for online evaluation of RL4Rec since it can indicate the long-term user engagement performance achieved by RL4Rec.

Release of implementation. The complete implementation of our experiments with accompanying documentation and additional resources are publicly available for future reproducibility at `https://github.com/BetsyHJ/RL4Rec`.

5.7 Experimental Results and Analysis

Our experiments results are meant to determine whether the main finding of Liu et al. [83] can be reproduced:

> *The attention state encoder for RL4Rec provides significantly higher performance than the BOI, PLD and Avg state encoders.*

Moreover, in our analysis we investigate whether this finding generalizes in the four directions described at the start of Section 5.5.

5.7.1 Comparison of State Encoders on Debiased Simulation of the Yahoo!R3 Dataset

We start our analysis by considering our first research question RQ7.1: *whether Liu et al. [83]'s finding generalizes to DQN-based RL4Rec methods when evaluated in the debiased SOFA simulator and compared with more state encoders*.

Figure 5.2a (top) displays the evaluation performance of the optimized policies based on four state encoders proposed by Liu et al. [83]; the reported metric is the average cumulative number of clicks received over 10 interaction turns. The first interaction turn is always represented by the empty state, and as a result, the choice of state encoder is inconsequential and the performance of all state encoders is identical. As the number of interaction turns becomes larger, the differences between the state encoders become more apparent. On the simulations of Yahoo!R3 dataset – the same dataset used by Liu et al. –, as shown in Figure 5.2a (top), we see results consistent with those reported by Liu et al.: (1) BOI and PLD perform comparably and worse than Avg; and (2) on average the attention state encoder outperforms BOI, PLD, and Avg.

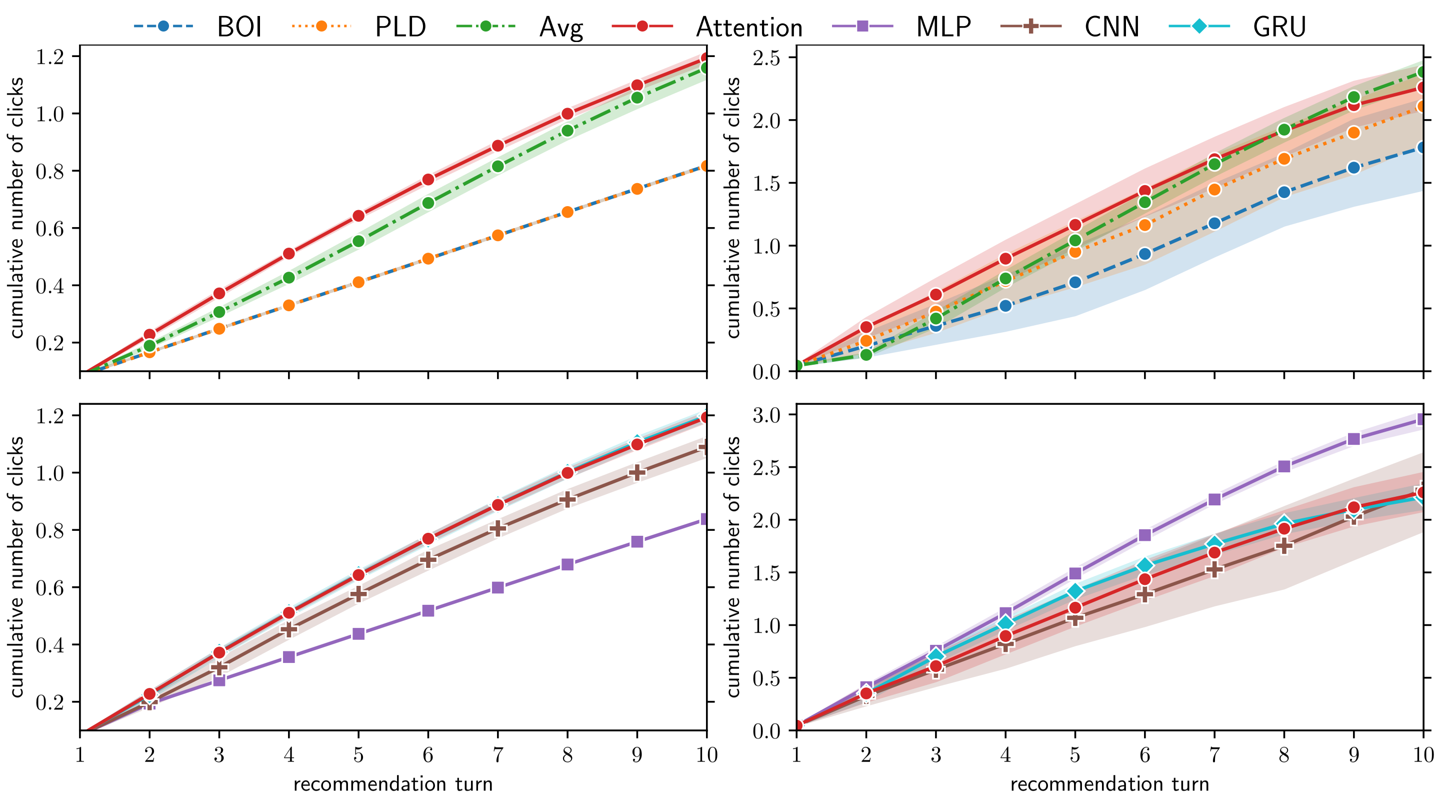
Figure 5.2a (bottom) displays the evaluation performance of the attention, MLP,[1] CNN and GRU state encoders on the Yahoo!R3 dataset. We see that on average the attention state encoder performs similarly to the GRU state encoder over ten interaction turns and better than the CNN state encoder. Thus, we confirm that attention is the optimal choice in our experimental setting on Yahoo!R3, the same dataset as used in the original comparison [83]. The main difference between MLP and GRU is the recurrent nature of the latter, thus it is the likely reason for why GRU outperforms MLP. Similarly, the higher performance of attention over GRU must be because of the additional attention layer, as this is the sole difference between the two state encoders.

We answer RQ7.1 in the affirmative: Liu et al.'s finding regarding the superiority of using the attention state encoder generalizes to DQN-based RL4Rec methods when evaluating in the debiased SOFA simulation based on the Yahoo!R3 dataset used by Liu et al., and compared with three more state encoders, MLP, GRU, and CNN.

5.7.2 Comparison on a Different Dataset

Now that we have found Liu et al.'s finding to be reproducible in a debiased simulation generated from the Yahoo!R3 dataset, we consider the second research question RQ7.2:

[1] For the MLP-based state encoder, we use a ReLU for Yahoo!R3 and tanh for the Coat shopping dataset, which we found to be the optimal choices for the corresponding datasets.



(a) Yahoo!R3.

(b) Coat shopping.

Figure 5.2: Comparisons of evaluation performance (the cumulative number of clicks) among policies with four state encoders proposed by Liu et al. [83] (top), and between attention and the additional MLP,[1] GRU, and CNN-based state encoders (bottom) on the unbiased simulations generated from the unbiased test sets of Yahoo!R3 and Coat shopping datasets, respectively.

whether it also generalizes to a debiased simulation based on a different dataset.

Figure 5.2b displays the performance of different state encoders on the debiased simulation based on the Coat shopping dataset, which was not part of the original comparison [83]. We make two observations from the top part of Figure 5.2b: (1) on average, PLD performs better than BOI, but worse than Avg; (2) attention has worse performance than Avg over 10 interaction turns. Furthermore, in the bottom part of Figure 5.2b we see that: (3) attention does not have better performance than the additional MLP, CNN and GRU state encoders; (4) on average, attention performs comparably with GRU and CNN, although CNN does suffer from a much higher variance; (5) the MLP state encoder outperforms other state encoders significantly. Thus, in stark contrast with our results on the Yahoo!R3 dataset, on the Coat shopping dataset we do not observe the attention state encoder to have the highest performance.

Two potential reasons for this observed inconsistency between the two datasets could be (1) the difference in size between the two datasets: in contrast to attention, the Avg and MLP methods with fewer parameters are possibly more effective on the smaller Coat shopping dataset; and (2) the different recommendation scenarios: there could be a stronger dependency between items in user interactions in an online shopping scenario (Coat shopping) than in a music recommendation scenario (Yahoo!R3).

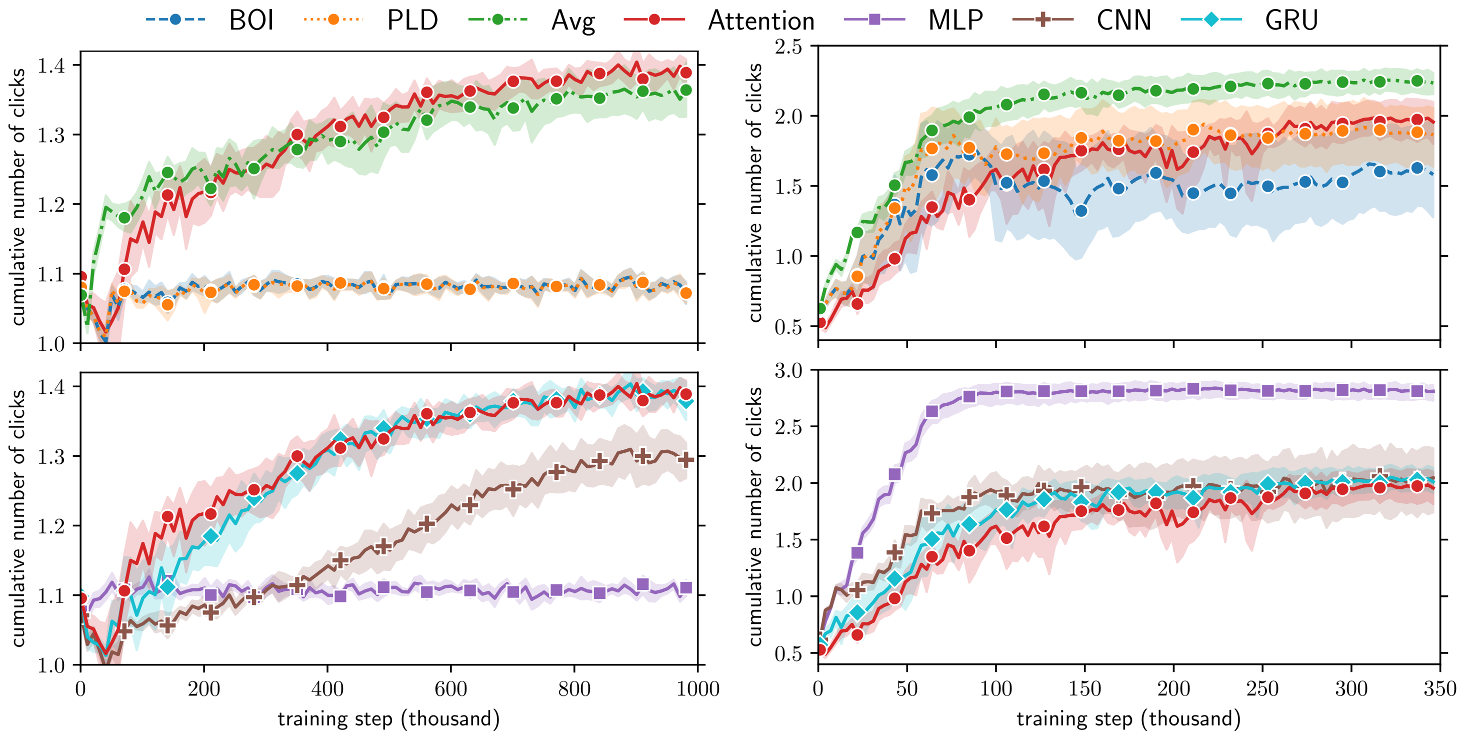
Therefore, we answer RQ7.2 negatively: Liu et al.'s finding does *not* generalize to the debiased simulation with a different dataset. In particular, attention is *not* the optimal choice of state encoder for RL4Rec when evaluating in the Coat shopping dataset, which was not considered by Liu et al. [83].

In addition, we also did not observe a consistent performance for the additional MLP, CNN and GRU state encoders across the two datasets. On the Yahoo!R3 dataset, GRU performs best (out of the three) and MLP performs worst; yet on the Coat shopping dataset GRU performs similarly to CNN but considerably worse than MLP. This observation suggests that the relative effectiveness of state encoders depends on the dataset to which they are applied. Importantly, there is no single optimal state encoder applicable to the RL4Rec method for all datasets.

5.7.3 Convergence of RL4Rec State Encoders

Convergence is also a crucial property for RL4Rec methods because they are more prone to divergence problems as they continuously update recommendation policies while interacting with users. Next, we investigate how the choice of state encoder affects the convergence of DQN, in terms of the number of training steps and training time needed to converge.

Figure 5.3 displays the learning curves of policies with different state encoders, which track the average cumulative number of clicks over 10 interaction turns on the debiased simulations on the Yahoo!R3 and the Coat shopping datasets. We observe that: (1) BOI and PLD converge the earliest but to policies that receive only a small cumulative number of clicks; (2) MLP has a similar convergence speed as BOI and PLD but its performance at convergence greatly varies between the datasets; (3) MLP converges faster than Avg, suggesting that its activation function speeds up the learning process; (4) attention converges slightly slower than GRU, most likely due to having more parameters; and (5) the convergence speed of CNN greatly varies between the two different datasets. In summary,



(a) Yahoo!R3.

(b) Coat shopping.

Figure 5.3: Learning curves tracking average cumulative number of clicks received by policies with four state encoders (top), and with attention and additional MLP, GRU and CNN-based state encoders (bottom) on the debiased simulations generated from training sets of Yahoo!R3 and Coat shopping datasets, respectively.

Table 5.2: Training time in seconds for 1,000 training steps.

Dataset	**BOI**	**PLD**	**Avg**	**MLP**	**CNN**	**GRU**	**Att**
Coat	6.6	7.0	6.4	6.4	7.6	23.4	25.4
Yahoo!R3	9.0	9.6	8.4	9.4	10.8	26.8	30.4

state encoders with few parameters, *e.g.,* Avg and MLP, converge faster than those with more parameters, *e.g.,* attention.

Furthermore, Table 5.2 clearly shows that the time for training on the larger Yahoo!R3 dataset is longer than on the Coat shopping dataset, which contains fewer items and users. As expected, Avg and MLP have the fewest parameters and accordingly also require less training time per thousand training steps. BOI and PLD take slightly more time than Avg which could be explained by the higher dimensionality of their state representations. Lastly, attention is more time-consuming than GRU, which is likely due to its additional attention layer. In summary, the attention state encoders require a higher computation cost, despite the fact that they do not always guarantee to reach the highest performance, *e.g.,* on the Coat shopping dataset.

5.7.4 Choice of Activation Functions for MLP

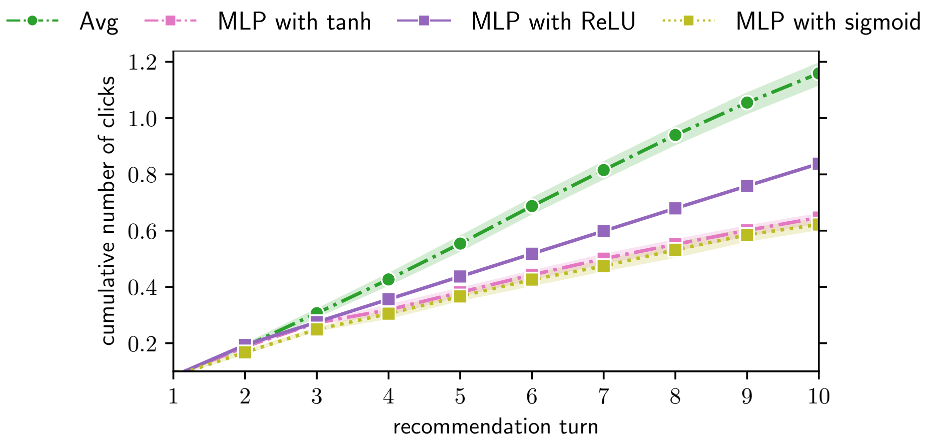
The MLP state encoders apply a non-linear activation function on top of Avg and show varying evaluation performance when applied to different datasets: we have seen that it performs best on the Coat shopping dataset, but worse than Avg on Yahoo!R3, as shown in Figure 5.2 (bottom row). These observations prompt us to consider RQ7.3: *whether the choice of activation functions should be taken into account when using the MLP-based state encoder for RL4Recs.*

Figure 5.4 displays the comparison of evaluation performance between Avg and MLPs with the tanh, ReLU, and sigmoid activation functions. We observe that: (1) interestingly, MLPs perform better on the Yahoo!R3 dataset but worse than Avg on the Coat shopping dataset; we speculate that this is due to the different sizes of the two datasets and the different recommendation scenarios they represent; (2) for MLPs, sigmoid is the worst choice of activation function for simulations on both datasets, probably because it is more prone to the vanishing gradient problem [11]; and (3) the performance with tanh and ReLU is not consistent across both datasets: tanh has the best performance on Coat, but is worse than ReLU on Yahoo!R3.

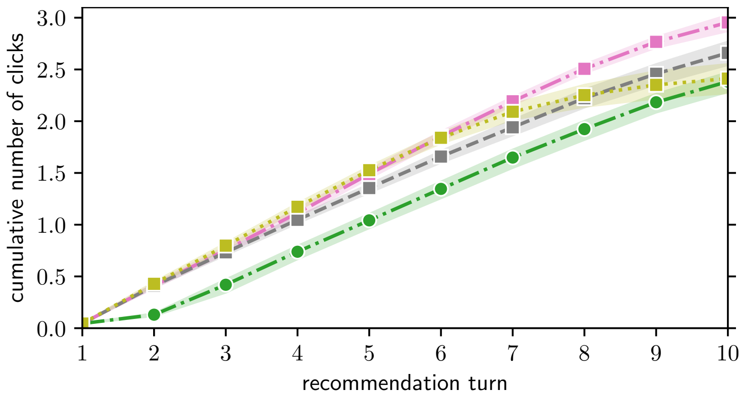
Therefore, we answer RQ7.3 in the affirmative: the choice of activation function should certainly be taken into account when using MLP-based state encoders for RL4Rec. Furthermore, our observations also suggest that the choice of activation function greatly depends on the dataset to which the MLP state encoder will be applied.

5.8 Conclusion

In this paper, we have reproduced and generalized a previous study by Liu et al. [83] regarding the choice of state encoder for reinforcement learning for recommenda-



(a) Yahoo!R3.



(b) Coat shopping.

Figure 5.4: Evaluation performance (the cumulative number of clicks) of policies with Avg, and MLP state encoders with different activation functions (tanh, ReLU, and sigmoid) on the simulations of Yahoo!R3 and Coat shopping datasets, respectively.

tions (RL4Recs) in four directions: (1) a debiased simulated environment, named SOFA; (2) RL4Rec methods based on Deep Q-Network (DQN), the most popular RL method used in RL4Recs; (3) three additional state encoders based on three typical neural networks: multi-layer perceptrons (MLPs), gated recurrent units (GRUs), and convolutional neural networks (CNNs); (4) besides the Yahoo!R3 dataset used in the original study [83], we also considered the Coat shopping dataset as the basis for debiased simulations. Our experimental results show that the higher performance of the attention state encoder over the bag of items (BOI), pairwise local dependency between items (PLD), and average (Avg) state encoders is reproducible in the debiased simulation generated from the Yahoo! R3 dataset, where DQN was used instead of actor-critic RL; moreover, the attention state encoder also outperforms the three additional multi-layer perceptron (MLP), convolutional neural network (CNN) and gated recurrent unit (GRU) state encoders on the debiased simulation based on the Yahoo!R3 dataset. However, the attention state encoder performed worse than Avg and MLP when the simulation is based on the Coat shopping dataset, a dataset not used in [83], despite the fact that it has the highest computational costs.

In summary, our results confirm that Liu et al.'s finding generalizes in the first three directions, *i.e.,* the debiased simulation, DQN-based RL4Rec method, and more state encoders, but does *not* generalize to the debiased simulation generated from a different dataset, *i.e.,* the Coat shopping dataset. In addition, we have found that the choice of activation function plays a crucial role when constructing a state encoder for RL4Rec. These findings allow us to answer the thesis-level research question RQ7 negatively: Liu et al. [83]'s finding, which suggests that attention is the optimal choice of state encoders for RL4Rec methods, does not generalize to the debiased simulation from the Coat shopping dataset.

Future work should further investigate the importance of the choice of RL methods for RL4Rec. A comparison of different RL methods, such as DQN, REINFORCE and actor-critic, in various RL4Rec frameworks could reveal whether comparisons of RL methods generalize across different settings. The resulting insights could greatly aid researchers and practitioners in the RL4Rec domain.

Implementation Resources and Data

To facilitate the reproducibility of the reported results, this study only made use of publicly available data. Our complete experimental implementation is publicly available with detailed instructions for reproducing our experiments at `https://github.com/BetsyHJ/RL4Rec`.

6 Conclusions

Debiasing recommender systems (RSs) is a crucial endeavor aimed at mitigating the negative effect of bias present in logged user interactions on RS methods that learn from these biased interactions. This mitigation of bias contributes to positive social impacts and creates more equitable RSs. This thesis has focused on two important topics in this area: (1) correcting for complex forms of selection bias, and (2) learning and evaluating reinforcement learning for recommendation (RL4Rec) methods in a debiased simulator.

In this final chapter, we revisit the thesis research questions that have been introduced in Section 1.1 and addressed by the following research chapters and provide a summary of the main findings. Finally, we discuss the potential future research directions that can build upon the work present in this thesis.

6.1 Main Findings

6.1.1 Correcting for Complex Forms of Selection Bias

The first part of the thesis looked at and corrected for two complex forms of selection bias. Chapter 2 considered a dynamic scenario in which both the selection bias and user preferences are dynamic and asked:

RQ1 Do we find evidence for dynamic selection bias and dynamic user preferences in real-world data?

We conducted a comparative analysis between methods that incorporate the item-age factor and those that do not, in predicting selection bias and user preferences within users' behavior in the MovieLens dataset. The experimental results show that the inclusion of the item-age factor is crucial, as it significantly improves the prediction performances. These findings strongly indicate that dynamic scenarios better capture user preferences in the real world. Consequently, we affirmatively answer this question: dynamic scenarios exist in the real world.

In order to mitigate the effect of bias on the resulting RSs in such dynamic scenarios, Chapter 2 also addressed the question:

RQ2 Can the prevalent IPS-based debiasing method be extended to mitigate the effect of dynamic user selection bias and model dynamic user preferences?

We answer this question positively by introducing DANCER, a method for DebiAsing in the dyNamiC scEnaRio that extends the IPS-based debiasing method by utilizing propensities that vary depending on the item-age. Our experimental results show that DANCER significantly outperforms the static IPS debiasing method that incorrectly assumes static selection bias in a dynamic scenario. We applied the proposed DANCER to a time-aware matrix factorization model, resulting in TMF-DANCER, the first recommendation method that corrects for dynamic selection bias and models dynamic user preferences.

Our findings about the dynamic scenario have implications for state-of-the-art recommendation methods, as they are strongly affected by dynamic selection bias. With the proposed DANCER debiasing methods, RS methods can now be expanded to deal with dynamic scenarios.

Chapter 3 looked at a multifactorial bias that is determined by the item and rating value factors. To mitigate the effect of multifactorial bias, Chapter 3 asked the question:

RQ3 Can the IPS-based debiasing method be extended to correct for multifactorial bias?

We proposed a propensity estimation method for multifactorial bias that considers both the item and rating value factors. Ideally, we expect that a rating prediction method optimized with IPS corrects for multifactorial bias if it utilizes the results of our multifactorial bias propensity estimation. However, it poses a severe sparsity problem due to the consideration of multiple factors. To make our proposed multifactorial method feasible and robust in practice, we also addressed the following question in Chapter 3:

RQ4 Can we deal with the severe sparsity problem posed by the multifactorial method?

We answer this question affirmatively by proposing the adoption of a propensity smoothing technique and a novel alternating gradient descent approach in our multifactor method. Our experimental results show that our proposed multifactorial method with adopting propensity smoothing and alternating gradient descent optimization effectively and robustly improves the rating prediction performance. Accordingly, we can answer RQ3 positively: upon resolving the sparsity issue, the IPS-based debiasing method can be extended to correct for multifactorial bias by using the proposed multifactorial bias propensity estimation.

As multifactorial bias appears to better capture real-world forms of bias, our proposed multifactorial debiasing approach makes a significant contribution to the RS field: its integration serves to significantly enhance the performance of RS methods when learning from biased user ratings.

6.1.2 Learning and Evaluating RL4Rec in a Debiased Simulator

The second part of the thesis considered the effect of bias present in logged user interactions on the resulting RL4Rec methods. Chapter 4 focused on debiasing simulators for RL4Rec and asked the question:

RQ5 Is it possible to mitigate the effect of bias on simulators for RL4Rec?

We proposed an intermediate bias mitigation step (IBMS) between the logged data and the learned rating prediction model in RL4Rec simulators. By mitigating the bias originating from the data from affecting the model to predict the user-item rating matrix, it minimizes the effect of bias on subsequent steps, including the simulated user behavior and the final produced RL4Rec method. To evaluate how well the proposed debiasing method mitigates the effect of bias, Chapter 4 also posed the following question:

RQ6 Can the evaluation of a simulator take the performance of the RL4Rec methods that are learned with this simulator into account?

We answer this question in the affirmative by proposing a novel evaluation approach. The proposed evaluation approach involves comparing the performance of a policy trained in a simulator with and without the IBMS. To conduct this evaluation, we developed an unbiased simulator based on a small set of user interactions on uniformly randomly selected items. This unbiased simulator serves to evaluate policy performance. We further presented two solutions to address the sparsity problem arising from using this small set of unbiased user interactions in constructing unbiased simulator. Using our proposed evaluating approach, our experimental results reveal that: (1) bias present in logged data affects a simulator, and (2) our proposed IBMS can mitigate the bias. Accordingly, we answer RQ5 in the affirmative: the effect of bias on simulators for RL4Rec can be mitigated by the proposed IBMS. Furthermore, we combine both the proposed debiasing method and evaluation approach in our newly proposed Simulator for OFfline leArning and evaluation (SOFA). We made our SOFA publicly available to help researchers in the field develop and evaluate RL4Rec methods while mitigating the effect of bias.

Our findings regarding the effect of bias on RL4Rec simulators have implications for the state-of-the-art RL4Rec methods that learn by using simulators that ignore bias present in logged user data. By training with the introduced SOFA simulator, the resulting RL4Rec methods suffer less from bias.

In Chapter 4, we have shown that bias present in logged user interactions negatively affects RL4Rec simulators and the resulting RL4Rec methods. Naturally, this leads to a question:

RQ7 Can the findings regarding the optimal choice of state encoders in RL4Rec methods generalize to the debiased simulation?

In Chapter 5, we answer this question by reproducing and expanding on the existing comparison of the optimal choice of state encoders, *i.e.,* the attention-based state encoder, in the publicly available debiased SOFA simulator introduced in Chapter 4. Moreover, we also considered other three generalization directions: a different RL method, more state encoders, and a different dataset. Our experimental results show that existing findings do *not* generalize to the debiased SOFA simulator generated from a different dataset and a Deep Q-Network (DQN)-based method when compared with more state encoders.

Our findings concerning the optimal choice of state encoder have implications on state-of-the-art RL4Rec methods, as state encoder plays a crucial role when constructing RL4Rec methods. Notably, the optimal choice of state encoders for RL4Rec methods may differ depending on whether a simulator ignores bias or corrects for bias.

6.2 Future Work

We conclude this thesis by suggesting promising research directions for future work.

Addressing complex forms of bias in general RS scenarios. We extended the prevalent IPS-based debiasing method to correct for two complex forms of bias in the context of the rating prediction task in the first part of this thesis. Our experimental results demonstrated the effectiveness of the proposed debiasing methods in improving rating prediction performance. One natural extension direction is to generalize the proposed debiasing methods to item recommendation tasks, such as sequential recommendations. Unlike the rating prediction task, where the goal is to predict user ratings on items according to explicit historical user feedback, the item recommendation task aims to recommend a list of items to users based on user historical implicit feedback, *e.g.,* clicks. The task of sequential recommendations further involves dynamic short-term user preferences [35]. Ideally, the proposed debiasing methods, which are the extended variant of the IPS-based debiasing method, apply equally to the item recommendation task as to the rating prediction task [62, 117]. However, methods for sequential recommendation are often very complex, *e.g.,* based on recurrent neural network [48] or BERT [128]. These complex methods inherently introduce variance issues and exacerbate the issues when combined with the proposed debiasing methods. Hence, the challenge for future work on debiasing sequential recommendation is to strike a balance between effectively addressing bias and reducing variance resulting from the combination of complex sequential recommendation methods and debiasing methods.

Besides debiasing in the context of item recommendation tasks, one other promising direction is to provide a general propensity score learning algorithm and integrate it with the IPS debiasing methods to deal with the simultaneous occurrence of various biases in real-world scenarios. Chapter 3 is a preliminary attempt to successfully correct for multifactorial bias, which can be seen as the combination of popularity bias and positivity bias [55]. Given the inherent complexity of user behavior in practice, it is evident that we must identify and correct for complex bias scenarios [21]. Interestingly, the observed distribution shift between natural user interactions and user interactions on uniformly randomly selected items may not always be attributed to bias. For instance, a long-tailed rating distribution, commonly referred to as the evidence of popularity bias, could result from positivity bias if there are only a few items with high rating values then these items will get the most ratings due to positivity bias as discussed in Chapter 3 [55]. Moreover, as Knyazev and Oosterhuis [66] have pointed out, the bandwagon effect, where user feedback is often influenced by earlier interactions of other users, is not a problem of statistical bias. Hence, we recognize a two-fold challenge for future work: distinguish genuine bias-related shifts from other factors affecting the data distribution and find a universal method that can automatically detect various types of bias and effectively correct for them.

Improving reinforcement learning for recommendation. In Chapter 4, we introduced a debiased simulator SOFA to mitigate the effect of bias present in logged user interactions on the resulting RL4Rec methods. The proposed debiased SOFA simulator only simulates the static user preference and the single-item recommendation scenario. One evident and valuable extension of this work involves developing a simulator that considers both static long-term user preferences and dynamic short-term user preferences. Similar to SOFA, this extended simulator also needs to deal with bias present in logged user interactions, relying on the development of debiasing sequential recommendations as introduced above. Moreover, it is more common to recommend a list of items or a slate of items to users than a single item [31, 163]. In these scenarios, user behavior on an item might be affected by the position of the item and the surrounding items [96, 115, 169]. One challenge to future work is to simulate user behavior towards multiple items on a recommendation page with taking user preferences and the positions of items into account.

In addition, we conducted comparisons of different state encoders for a DQN-based RL4Rec method using the proposed debiased simulator in Chapter 5. Future work should investigate the importance of the choice of RL methods, *e.g.,* REINFORCE [141] and actor-critic [76], when building RL4Rec methods. Besides various RL methods, various reward functions have been proposed for RL4Rec methods to guide the methods towards achieving desired outcomes [4]. In fact, the learning of RL4Rec methods solely relies on the reward signal. It is imperative to investigate the optimal choice of reward functions for RL4Rec methods in future work. The primary challenges in this line of research involve ensuring the efficiency of reward functions in the context of dealing with large-scale recommender systems and achieving broader goals that extend beyond traditional accuracy measurements, *e.g.,* promoting fairness and divergence in recommendations.

Overall, my main suggestions for future work are to focus on the complexities of real-world user behavior and to reduce the high variance that complex recommendation methods produce when mitigating the effect of bias.

Bibliography

[1] H. Abdollahpouri. Popularity Bias in Ranking and Recommendation. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 529–530. ACM, 2019. (Cited on page 36.)

[2] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher. The Connection between Popularity Bias, Calibration, and Fairness in Recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 726–731. ACM, 2020. (Cited on page 2.)

[3] P. Adamopoulos and A. Tuzhilin. On Over-Specialization and Concentration Bias of Recommendations: Probabilistic Neighborhood Selection in Collaborative Filtering Systems. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 153–160. ACM, 2014. (Cited on pages 2, 14, 34, and 80.)

[4] M. M. Afsar, T. Crump, and B. Far. Reinforcement Learning based Recommender Systems: A Survey. *ACM Computing Surveys*, 55(7):1–38, 2022. (Cited on pages 1, 77, 79, and 101.)

[5] I. A. A.-Q. Al-Hadi, N. M. Sharef, M. N. Sulaiman, and N. Mustapha. Review of the Temporal Recommendation System with Matrix Factorization. *Int. J. Innov. Comput. Inf. Control*, 13(5): 1579–1594, 2017. (Cited on page 14.)

[6] R. Baeza-Yates. Bias on the Web. *Communications of the ACM*, 61(6):54–61, 2018. (Cited on pages 17 and 49.)

[7] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations*, 2015. (Cited on page 87.)

[8] L. Baltrunas and X. Amatriain. Towards Time-Dependant Recommendation based on Implicit Feedback. In *Workshop on context-aware recommender systems (CARS'09)*, pages 25–30. Citeseer, 2009. (Cited on page 16.)

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. (Cited on page 62.)

[10] A. Bellogín, P. Castells, and I. Cantador. Statistical Biases in Information Retrieval Metrics for Recommender Systems. *Information Retrieval Journal*, 20:606–634, 2017. (Cited on page 48.)

[11] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994. (Cited on page 93.)

[12] L. Bernardi, S. Batra, and C. A. Bruscantini. Simulations in Recommender Systems: An Industry Perspective. *arXiv preprint arXiv:2109.06723*, 2021. (Cited on pages 5, 78, and 80.)

[13] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender Systems Survey. *Knowledge-Based Systems*, 46:109–132, 2013. (Cited on pages 1 and 34.)

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016. (Cited on page 62.)

[15] B. Brost, R. Mehrotra, and T. Jehan. The Music Streaming Sessions Dataset. In *The World Wide Web Conference*, pages 2594–2600. ACM, 2019. (Cited on page 26.)

[16] R. Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-adapted Interaction*, 12:331–370, 2002. (Cited on page 34.)

[17] P. G. Campos, F. Díez, and I. Cantador. Time-Aware Recommender Systems: A Comprehensive Survey and Analysis of Existing Evaluation Protocols. *User Modeling and User-Adapted Interaction*, 24(1): 67–119, 2014. (Cited on pages 15, 16, and 22.)

[18] R. Cañamares and P. Castells. Should I Follow the Crowd?: A Probabilistic Analysis of the Effectiveness of Popularity in Recommender Systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 415–424. ACM, 2018. (Cited on pages 2, 3, 14, 20, 26, 34, 35, 50, and 57.)

[19] A. Chakraborty, S. Ghosh, N. Ganguly, and K. P. Gummadi. Optimizing the Recency-Relevancy Trade-off in Online News Recommendations. In *Proceedings of the 26th International Conference on World Wide Web*, pages 837–846. ACM, 2017. (Cited on pages 3 and 14.)

[20] H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu. Large-Scale Interactive Recommendation with Tree-structured Policy Gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3312–3320, 2019. (Cited on pages 58 and 61.)

[21] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, and X. He. Bias and Debias in Recommender System: A Survey and Future Directions. *ACM Transactions on Information Systems*, 41(3):1–39, 2023. (Cited on pages 2, 3, 14, 16, 34, 36, 49, 50, 80, and 100.)

[22] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi. Top-k Off-policy Correction for a REINFORCE Recommender System. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464. ACM, 2019. (Cited on pages 50, 58, 62, 79, and 80.)

[23] R.-C. Chen, Q. Ai, G. Jayasinghe, and W. B. Croft. Correcting for Recency Bias in Job Recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2185–2188. ACM, 2019. (Cited on pages 14, 16, 62, and 80.)

[24] S.-Y. Chen, Y. Yu, Q. Da, J. Tan, H.-K. Huang, and H.-H. Tang. Stabilizing Reinforcement Learning in Dynamic Environment with Application to Online Recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1187–1196. ACM, 2018. (Cited on pages 56, 61, 78, 79, and 85.)

[25] X. Chen, H. Xu, Y. Zhang, J. Tang, Y. Cao, Z. Qin, and H. Zha. Sequential Recommendation with User Memory Networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 108–116. ACM, 2018. (Cited on pages 15 and 80.)

[26] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System. In *International Conference on Machine Learning*, pages 1052–1061, 2019. (Cited on pages 6, 65, 78, 79, and 85.)

[27] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10. ACM, 2016. (Cited on pages 15 and 80.)

[28] S. Choi, H. Ha, U. Hwang, C. Kim, J.-W. Ha, and S. Yoon. Reinforcement Learning Based Recommender System Using Biclustering Technique. *arXiv preprint arXiv:1801.05532*, 2018. (Cited on page 61.)

[29] G. L. Ciampaglia, A. Nematzadeh, F. Menczer, and A. Flammini. How Algorithmic Popularity Bias Hinders or Promotes Quality. *Scientific Reports*, 8(1):1–7, 2018. (Cited on page 20.)

[30] J. F. G. da Silva, N. N. de Moura Junior, and L. P. Caloba. Effects of Data Sparsity on Recommender Systems Based on Collaborative Filtering. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018. (Cited on pages 5 and 34.)

[31] R. Deffayet, T. Thonet, J.-M. Renders, and M. de Rijke. Generative Slate Recommendation with Reinforcement Learning. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 580–588. ACM, 2023. (Cited on page 101.)

[32] Y. Deldjoo, M. Schedl, P. Cremonesi, and G. Pasi. Recommender Systems Leveraging Multimedia Content. *ACM Computing Surveys*, 53(5):1–38, 2020. (Cited on page 1.)

[33] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. *arXiv preprint arXiv:1512.07679*, 2015. (Cited on pages 61 and 79.)

[34] M. D. Ekstrand, M. Tian, I. M. Azpiazu, J. D. Ekstrand, O. Anuyah, D. McNeill, and M. S. Pera. All the Cool Kids, How Do They Fit In?: Popularity and Demographic Biases in Recommender Evaluation and Effectiveness. In *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, pages 172–186. PMLR, 2018. (Cited on pages 2, 4, 34, and 39.)

[35] H. Fang, D. Zhang, Y. Shu, and G. Guo. Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations. *ACM Transactions on Information Systems*, 39(1):1–42, 2020. (Cited on page 100.)

[36] C. Gao, S. Li, W. Lei, J. Chen, B. Li, P. Jiang, X. He, J. Mao, and T.-S. Chua. KuaiRec: A Fully-observed Dataset and Insights for Evaluating Recommender Systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 540–550. ACM, 2022. (Cited on page 47.)

[37] A. Gilotte, C. Calauzènes, T. Nedelec, A. Abraham, and S. Dollé. Offline A/B Testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 198–206. ACM, 2018. (Cited on page 56.)

[38] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Inf. Retr.*, 4(2):133–151, 2001. (Cited on page 83.)

[39] C. Greco, A. Suglia, P. Basile, and G. Semeraro. Converse-Et-Impera: Exploiting Deep Learning and Hierarchical Reinforcement Learning for Conversational Recommender Systems. In *AI*IA 2017 Advances in Artificial Intelligence - XVIth International Conference of the Italian Association for Artificial Intelligence*, pages 372–386. Springer, 2017. (Cited on page 79.)

[40] S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Recent Advances in the Foundations and Applications of Unbiased Learning to Rank. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3440–3443. ACM, July 2023.

[41] S. Hajian, F. Bonchi, and C. Castillo. Algorithmic Bias: From Discrimination Discovery to Fairness-aware Data Mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2125–2126. ACM, 2016. (Cited on pages 17, 49, and 80.)

[42] P. Han, S. Shang, A. Sun, P. Zhao, K. Zheng, and P. Kalnis. AUC-MF: Point of Interest Recommendation

with AUC Maximization. In *2019 IEEE 35th International Conference on Data Engineering*, pages 1558–1561. IEEE, 2019. (Cited on page 15.)

[43] F. M. Harper and J. A. Konstan. The Movielens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4):1–19, 2015. (Cited on pages 4, 21, 22, and 83.)

[44] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural Collaborative Filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017. (Cited on pages 15 and 80.)

[45] X. He, X. Du, X. Wang, F. Tian, J. Tang, and T.-S. Chua. Outer Product-based Neural Collaborative Filtering. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2227–2233, 2018. (Cited on pages 15 and 80.)

[46] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 639–648. ACM, 2020. (Cited on page 15.)

[47] J. M. Hernández-Lobato, N. Houlsby, and Z. Ghahramani. Probabilistic Matrix Factorization with Non-Random Missing Data. In *International Conference on Machine Learning*, pages 1512–1520. JMLR, 2014. (Cited on pages 16, 50, and 62.)

[48] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations*, 2016. (Cited on pages 1, 15, 80, and 100.)

[49] D. G. Horvitz and D. J. Thompson. A Generalization of Sampling without Replacement from a Finite Universe. *Journal of the American statistical Association*, 47(260):663–685, 1952. (Cited on page 80.)

[50] J. Huang, W. X. Zhao, H. Dou, J.-R. Wen, and E. Y. Chang. Improving Sequential Recommendation with Knowledge-enhanced Memory Networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 505–514. ACM, June 2018. (Cited on pages 15 and 80.)

[51] J. Huang, Z. Ren, W. X. Zhao, G. He, J.-R. Wen, and D. Dong. Taxonomy-Aware Multi-Hop Reasoning Networks for Sequential Recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 573–581. ACM, February 2019.

[52] J. Huang, H. Oosterhuis, M. de Rijke, and H. van Hoof. Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 190–199. ACM, September 2020. (Cited on pages 14, 34, 35, 38, 50, 78, 80, 81, 84, 85, and 88.)

[53] J. Huang, H. Oosterhuis, B. Cetinkaya, T. Rood, and M. de Rijke. State Encoders in Reinforcement Learning for Recommendation: A Reproducibility Study. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2738–2748. ACM, July 2022.

[54] J. Huang, H. Oosterhuis, and M. de Rijke. It Is Different When Items are Older: Debiasing Recommendations When Selection Bias and User Preferences are Dynamic. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 381–389. ACM, February 2022. (Cited on pages 42 and 50.)

[55] J. Huang, H. Oosterhuis, M. Mansoury, H. van Hoof, and M. de Rijke. Going Beyond Popularity and Positivity Bias: Correcting for Multifactorial Bias in Recommender Systems, August 2023. Under review. (Cited on page 100.)

[56] N. Idrissi, A. Zellou, and Z. Bakkoury. "Guess Why I Didn't Rate It": A New Preference-based Model for Enhanced Top-K Recommendation. *International Journal of Intelligent Engineering and Systems*, 16(3):542–551, 2023. (Cited on pages 2, 4, 34, and 39.)

[57] E. Ie, C.-w. Hsu, M. Mladenov, V. Jain, S. Narvekar, J. Wang, R. Wu, and C. Boutilier. RecSim: A Configurable Simulation Platform for Recommender Systems. *arXiv preprint arXiv:1909.04847*, 2019. (Cited on pages 5, 56, 62, 63, and 80.)

[58] E. Ie, V. Jain, J. Wang, S. Narvekar, R. Agarwal, R. Wu, H.-T. Cheng, M. Lustman, V. Gatto, P. Covington, et al. Reinforcement Learning for Slate-based Recommender Systems: A Tractable Decomposition and Practical Methodology. *arXiv preprint arXiv:1905.12767*, 2019. (Cited on pages 78, 79, and 85.)

[59] G. W. Imbens and D. B. Rubin. *Causal Inference in Statistics, Social, and Biomedical Sciences*. Cambridge University Press, 2015. (Cited on pages 2, 14, 17, 34, 38, 50, 62, 63, 64, and 84.)

[60] R. Jagerman, I. Markov, and M. de Rijke. When People Change their Mind: Off-policy Evaluation in Non-stationary Recommendation Environments. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 447–455. ACM, 2019. (Cited on pages 14 and 17.)

[61] Y. Ji, A. Sun, J. Zhang, and C. Li. A Re-visit of the Popularity Baseline in Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in*

Information Retrieval, pages 1749–1752. ACM, 2020. (Cited on pages 3, 14, and 16.)

[62] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased Learning-to-Rank with Biased Feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789. ACM, 2017. (Cited on pages 2, 3, 14, 16, 34, 38, 50, 62, 80, and 100.)

[63] J. D. Kang, J. L. Schafer, et al. Demystifying Double Robustness: A Comparison of Alternative Strategies for Estimating a Population Mean from Incomplete Data. *Statistical science*, 22(4):523–539, 2007. (Cited on page 80.)

[64] W.-C. Kang and J. McAuley. Self-Attentive Sequential Recommendation. In *IEEE International Conference on Data Mining*, pages 197–206. IEEE Computer Society, 2018. (Cited on page 1.)

[65] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015. (Cited on page 41.)

[66] N. Knyazev and H. Oosterhuis. The Bandwagon Effect: Not Just Another Bias. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 243–253. ACM, 2022. (Cited on page 100.)

[67] Y. Koren. The Bellkor Solution to the Netflix Grand Prize. *Netflix Prize Documentation*, 81(2009): 1–10, 2009. (Cited on page 15.)

[68] Y. Koren. Collaborative Filtering with Temporal Dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 447–456. ACM, 2009. (Cited on pages 15, 16, 20, and 22.)

[69] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, 2009. (Cited on page 15.)

[70] S. Krishnan, J. Patel, M. J. Franklin, and K. Goldberg. A Methodology for Learning, Analyzing, and Mitigating Social Influence Bias in Recommender Systems. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 137–144. ACM, 2014. (Cited on pages 2 and 50.)

[71] L. Li, W. Chu, J. Langford, and R. E. Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670. ACM, 2010. (Cited on pages 56 and 80.)

[72] L. Li, J. Y. Kim, and I. Zitouni. Toward Predicting the Outcome of an A/B Experiment for Search Relevance. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 37–46. ACM, 2015. (Cited on pages 1 and 56.)

[73] M. Li, J. Huang, and M. de Rijke. Repetition and Exploration in Offline Reinforcement Learning-based Recommendations, October 2023. DRL4IR workshop at CIKM 2023.

[74] H. Liang. Drprofiling: Deep Reinforcement User Profiling for Recommendations in Heterogenous Information Networks. *IEEE Transactions on Knowledge and Data Engineering*, 2020. (Cited on page 79.)

[75] E. Liebman, M. Saar-Tsechansky, and P. Stone. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 591–599. ACM, 2015. (Cited on page 1.)

[76] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous Control with Deep Reinforcement Learning. In *4th International Conference on Learning Representations*, 2016. (Cited on pages 78, 79, and 101.)

[77] Y. Lin, Y. Liu, F. Lin, P. Wu, W. Zeng, and C. Miao. A Survey on Reinforcement Learning for Recommender Systems. *arXiv preprint arXiv:2109.10665*, 2021. (Cited on page 77.)

[78] Y. Lin, Y. Liu, F. Lin, L. Zou, P. Wu, W. Zeng, H. Chen, and C. Miao. A Survey on Reinforcement Learning for Recommender Systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2023. (Cited on page 1.)

[79] G. Linden, B. Smith, and J. York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet computing*, 7(1):76–80, 2003. (Cited on page 1.)

[80] R. J. Little and D. B. Rubin. *Statistical Analysis with Missing Data*, volume 793. John Wiley & Sons, 2019. (Cited on pages 17 and 63.)

[81] D. Liu and C. Yang. A Deep Reinforcement Learning Approach to Proactive Content Pushing and Recommendation for Mobile Users. *IEEE Access*, 7:83120–83136, 2019. (Cited on pages 78, 79, and 85.)

[82] F. Liu, H. Guo, X. Li, R. Tang, Y. Ye, and X. He. End-to-End Deep Reinforcement Learning based Recommendation with Supervised Embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 384–392. ACM, 2020. (Cited on pages 78, 79, and 85.)

[83] F. Liu, R. Tang, X. Li, W. Zhang, Y. Ye, H. Chen, H. Guo, Y. Zhang, and X. He. State Representation Modeling for Deep Reinforcement Learning based Recommendation. *Knowledge-Based Systems*, 205: 106170, 2020. (Cited on pages 3, 6, 78, 79, 80, 81, 82, 83, 84, 86, 87, 88, 89, 90, 91, 93, 94, and 95.)

[84] J. Liu, P. Dolan, and E. R. Pedersen. Personalized News Recommendation based on Click Behavior.

In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, pages 31–40. ACM, 2010. (Cited on page 1.)

[85] S. Liu, Y. Chen, H. Huang, L. Xiao, and X. Hei. Towards Smart Educational Recommendations with Reinforcement Learning in Classroom. In *IEEE International Conference on Teaching, Assessment, and Learning for Engineering*, pages 1079–1084. IEEE, 2018. (Cited on pages 78, 79, and 85.)

[86] J. Ma, Z. Zhao, X. Yi, J. Yang, M. Chen, J. Tang, L. Hong, and E. H. Chi. Off-policy Learning in Two-stage Recommender Systems. In *Proceedings of The Web Conference 2020*, pages 463–473. ACM / IW3C2, 2020. (Cited on pages 79 and 80.)

[87] C. D. Manning, R. Prabhakar, and S. Hinrich. *Introduction to Information Retrieval*. Cambridge University Press, 2008. (Cited on page 40.)

[88] B. Marlin, R. S. Zemel, S. Roweis, and M. Slaney. Collaborative Filtering and the Missing at Random Assumption. *arXiv preprint arXiv:1206.5267*, 2012. (Cited on page 50.)

[89] B. M. Marlin and R. S. Zemel. Collaborative Prediction and Ranking with Non-random Missing Data. In *Proceedings of the Third ACM Conference on Recommender Systems*, pages 5–12. ACM, 2009. (Cited on pages 2, 6, 14, 16, 17, 26, 34, 42, 50, 57, 58, 59, 60, 62, 64, 67, 78, 83, and 87.)

[90] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney. Collaborative Filtering and the Missing at Random Assumption. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 267–275. AUAI Press, 2007. (Cited on page 62.)

[91] P. Massa and P. Avesani. Trust-Aware Recommender Systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007. (Cited on page 42.)

[92] D. F. McCaffrey, G. Ridgeway, and A. R. Morral. Propensity Score Estimation with Boosted Regression for Evaluating Causal Effects in Observational Studies. *Psychological Methods*, 9(4):403, 2004. (Cited on pages 2 and 50.)

[93] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys*, 54(6):1–35, 2021. (Cited on page 49.)

[94] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015. (Cited on pages 78 and 85.)

[95] T. T. Nguyen, P.-M. Hui, F. M. Harper, L. Terveen, and J. A. Konstan. Exploring the Filter Bubble: The Effect of Using Recommender Systems on Content Diversity. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 677–686. ACM, 2014. (Cited on pages 2, 14, 34, and 80.)

[96] H. Oosterhuis. *Learning from User Interactions with Rankings: A Unification of the Field*. PhD thesis, University of Amsterdam, 2020. (Cited on page 101.)

[97] H. Oosterhuis. Doubly Robust Estimation for Correcting Position Bias in Click Feedback for Unbiased Learning to Rank. *ACM Transactions on Information Systems*, 41(3):1–33, 2023. (Cited on page 50.)

[98] Z. Ovaisi, R. Ahsan, Y. Zhang, K. Vasilaky, and E. Zheleva. Correcting for Selection Bias in Learning-to-Rank Systems. In *Proceedings of The Web Conference 2020*, pages 1863–1873. ACM / IW3C2, 2020. (Cited on pages 2, 14, and 34.)

[99] F. Pan, Q. Cai, P. Tang, F. Zhuang, and Q. He. Policy Gradients for Contextual Recommendations. In *The World Wide Web Conference*, pages 1421–1431. ACM, 2019. (Cited on page 77.)

[100] U. Panniello, A. Tuzhilin, M. Gorgoglione, C. Palmisano, and A. Pedone. Experimental Comparison of Pre-vs. Post-filtering Approaches in Context-aware Recommender Systems. In *Proceedings of the Third ACM Conference on Recommender Systems*, pages 265–268. ACM, 2009. (Cited on page 16.)

[101] U. Panniello, S. Hill, and M. Gorgoglione. The Impact of Profit Incentives on the Relevance of Online Recommendations. *Electronic Commerce Research and Applications*, 20:87–104, 2016. (Cited on pages 2 and 50.)

[102] E. Pariser. *The Filter Bubble: How the New Personalized Web is Changing What We Read and How We Think*. Penguin, 2011. (Cited on pages 2, 14, 34, and 80.)

[103] M. J. Pazzani and D. Billsus. Content-Based Recommendation Systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 325–341. Springer, 2007. (Cited on page 1.)

[104] B. Pradel, N. Usunier, and P. Gallinari. Ranking with Non-random Missing Ratings: Influence of Popularity and Positivity on Evaluation Metrics. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 147–154. ACM, 2012. (Cited on pages 2, 4, 14, 16, 17, 24, 34, 35, 36, 39, 50, 57, 59, 60, 62, 78, 80, and 84.)

[105] F. Qi, X. Tong, L. Yu, and Y. Wang. Personalized Project Recommendations: Using Reinforcement Learning. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–17, 2019. (Cited on pages 78, 79, and 85.)

[106] M. Quadrana, P. Cremonesi, and D. Jannach. Sequence-Aware Recommender Systems. *ACM*

Computing Surveys, 51(4):1–36, 2018. (Cited on page 15.)

[107] P. Resnick and H. R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997. (Cited on page 1.)

[108] F. Ricci, L. Rokach, and B. Shapira. Introduction to Recommender Systems Handbook. *Recommender Systems Handbook*, pages 1–35, 2011. (Cited on pages 1, 5, and 34.)

[109] F. Ricci, L. Rokach, and B. Shapira. Recommender Systems: Introduction and Challenges. *Recommender Systems Handbook*, pages 1–34, 2015. (Cited on page 34.)

[110] J. M. Robins, A. Rotnitzky, and L. P. Zhao. Estimation of Regression Coefficients When Some Regressors are not Always Observed. *Journal of the American statistical Association*, 89(427):846–866, 1994. (Cited on page 80.)

[111] D. Rohde, S. Bonner, T. Dunlop, F. Vasile, and A. Karatzoglou. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. *arXiv preprint arXiv:1808.00720*, 2018. (Cited on pages 5, 56, 62, and 80.)

[112] P. R. Rosenbaum. Overt Bias in Observational Studies. In *Observational Studies*, pages 71–104. Springer, 2002. (Cited on pages 17 and 64.)

[113] Y. Saito, H. Sakata, and K. Nakata. Doubly Robust Prediction and Evaluation Methods Improve Uplift modeling for Observational Data. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 468–476. SIAM, 2019. (Cited on page 50.)

[114] Y. Saito, S. Yaginuma, Y. Nishino, H. Sakata, and K. Nakata. Unbiased Recommender Learning from Missing-Not-At-Random Implicit Feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 501–509. ACM, 2020. (Cited on pages 2, 20, 39, 42, and 50.)

[115] F. Sarvi, A. Vardasbi, M. Aliannejadi, S. Schelter, and M. de Rijke. On the Impact of Outlier Bias on User Clicks. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–27. Association for Computing Machinery, 2023. (Cited on pages 50 and 101.)

[116] H. Schäfer, S. Hors-Fraile, R. P. Karumur, A. Calero Valdez, A. Said, H. Torkamaan, T. Ulmer, and C. Trattner. Towards Health (Aware) Recommender Systems. In *Proceedings of the 2017 International Conference on Digital Health*, pages 157–161. ACM, 2017. (Cited on page 1.)

[117] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims. Recommendations as Treatments: Debiasing Learning and Evaluation. In *International Conference on Machine Learning*, pages 1670–1679. JMLR, 2016. (Cited on pages 2, 3, 6, 14, 16, 17, 26, 28, 34, 35, 37, 38, 40, 42, 50, 57, 58, 61, 62, 64, 66, 67, 68, 69, 78, 79, 80, 84, 85, 87, and 100.)

[118] B. Shi, M. G. Ozsoy, N. Hurley, B. Smyth, E. Z. Tragos, J. Geraci, and A. Lawlor. PyRecGym: A Reinforcement Learning Gym for Recommender Systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 491–495. ACM, 2019. (Cited on pages 56, 62, and 80.)

[119] J.-C. Shi, Y. Yu, Q. Da, S.-Y. Chen, and A.-X. Zeng. Virtual-taobao: Virtualizing Real-world Online Retail Environment for Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4902–4909. AAAI Press, 2019. (Cited on pages 6, 56, 62, 65, and 80.)

[120] T. Silveira, M. Zhang, X. Lin, Y. Liu, and S. Ma. How Good Your Recommender System Is? A Survey on Evaluations in Recommendation. *International Journal of Machine Learning and Cybernetics*, 10 (5):813–831, 2019. (Cited on page 71.)

[121] B. Smith and G. Linden. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Comput.*, 21(3):12–18, 2017. (Cited on page 1.)

[122] H. Steck. Training and Testing of Recommender Systems on Data Missing Not At Random. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 713–722. ACM, 2010. (Cited on pages 16, 57, and 80.)

[123] H. Steck. Item Popularity and Recommendation Accuracy. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, pages 125–132. ACM, 2011. (Cited on pages 2, 14, 16, 17, 34, 35, 36, 50, 57, 59, 62, 78, and 84.)

[124] H. Steck. Evaluation of Recommendations: Rating-prediction and Ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 213–220. ACM, 2013. (Cited on pages 16, 35, 42, 50, and 64.)

[125] A. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from Logged Implicit Exploration Data. In *Advances in Neural Information Processing Systems*, pages 2217–2225. Curran Associates, Inc., 2010. (Cited on page 39.)

[126] X. Su and T. M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009:421425:1–421425:19, 2009. (Cited on page 1.)

[127] V. Subramaniyaswamy, R. Logesh, M. Chandrashekhar, A. Challa, and V. Vijayakumar. A Personalised

Movie Recommendation System based on Collaborative Filtering. *International Journal of High Performance Computing and Networking*, 10(1-2):54–63, 2017. (Cited on page 1.)

[128] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450. ACM, 2019. (Cited on pages 1, 15, and 100.)

[129] Y. Sun and Y. Zhang. Conversational Recommender System. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 235–244. ACM, 2018. (Cited on page 79.)

[130] P. Sunehag, R. Evans, G. Dulac-Arnold, Y. Zwols, D. Visentin, and B. Coppin. Deep Reinforcement Learning with Attention for Slate Markov Decision Processes with High-Dimensional States and Actions. *arXiv preprint arXiv:1512.01124*, 2015. (Cited on pages 78, 79, and 85.)

[131] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. (Cited on pages 1, 56, 58, and 85.)

[132] A. Swaminathan and T. Joachims. The Self-Normalized Estimator for Counterfactual Learning. In *Advances in Neural Information Processing Systems*, pages 3231–3239. MIT Press, 2015. (Cited on pages 24 and 47.)

[133] G. Takács and D. Tikk. Alternating Least Squares for Personalized Ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, pages 83–90. ACM, 2012. (Cited on page 41.)

[134] P. B. Thorat, R. M. Goudar, and S. Barve. Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System. *International Journal of Computer Applications*, 110(4):31–36, 2015. (Cited on page 1.)

[135] T. N. T. Tran, A. Felfernig, C. Trattner, and A. Holzinger. Recommender Systems in the Healthcare Domain: State-of-the-art and Research Issues. *Journal of Intelligent Information Systems*, 57:171–201, 2021. (Cited on page 1.)

[136] M. Unger, A. Tuzhilin, and A. Livne. Context-Aware Recommendations Based on Deep Learning Frameworks. *ACM Trans. Manag. Inf. Syst.*, 11(2):1–15, 2020. (Cited on page 16.)

[137] L. Wang, W. Zhang, X. He, and H. Zha. Supervised Reinforcement Learning with Recurrent Neural Network for Dynamic Treatment Recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2447–2456. ACM, 2018. (Cited on pages 56 and 61.)

[138] X. Wang, R. Zhang, Y. Sun, and J. Qi. Doubly Robust Joint Learning for Recommendation on Data Missing Not At Random. In *International Conference on Machine Learning*, pages 6638–6647. PMLR, 2019. (Cited on pages 16, 26, 42, 50, and 62.)

[139] X. Wang, H. Jin, A. Zhang, X. He, T. Xu, and T.-S. Chua. Disentangled Graph Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1001–1010. ACM, 2020. (Cited on page 15.)

[140] C. Wangwatcharakul and S. Wongthanavasu. Dynamic Collaborative Filtering Based on User Preference Drift and Topic Evolution. *IEEE Access*, 8:86433–86447, 2020. (Cited on page 14.)

[141] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 8(3):229–256, 1992. (Cited on pages 79, 85, and 101.)

[142] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing. Recurrent Recommender Networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 495–503. ACM, 2017. (Cited on pages 15 and 80.)

[143] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan. Session-based Recommendation with Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 346–353. AAAI Press, 2019. (Cited on page 15.)

[144] X. Wu, H. Chen, J. Zhao, L. He, D. Yin, and Y. Chang. Unbiased Learning to Rank in Feeds Recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 490–498. ACM, 2021. (Cited on page 50.)

[145] Y. Xian, Z. Fu, S. Muthukrishnan, G. De Melo, and Y. Zhang. Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 285–294. ACM, 2019. (Cited on page 79.)

[146] X. Xin, X. Zhao, J. Huang, W. Zhang, L. Zhao, D. Yin, and G. H. Yang. DRL4IR: 4th Workshop on Deep Reinforcement Learning for Information Retrieval. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 5304–5307. ACM, October 2023.

[147] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell. Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization. In *Proceedings of the 2010 SIAM International*

Conference on Data Mining, pages 211–222. SIAM, 2010. (Cited on pages 15, 16, 20, and 22.)

[148] C. Xu, P. Zhao, Y. Liu, V. S. Sheng, J. Xu, F. Zhuang, J. Fang, and X. Zhou. Graph Contextualized Self-Attention Network for Session-based Recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3940–3946, 2019. (Cited on page 15.)

[149] L. Yang, Y. Cui, Y. Xuan, C. Wang, S. Belongie, and D. Estrin. Unbiased Offline Recommender Evaluation for Missing-Not-At-Random Implicit Feedback. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 279–287. ACM, 2018. (Cited on pages 24, 39, and 50.)

[150] S. Yao, Y. Halpern, N. Thain, X. Wang, K. Lee, F. Prost, E. H. Chi, J. Chen, and A. Beutel. Measuring Recommender System Effects with Simulated Users. *arXiv preprint arXiv:2101.04526*, 2021. (Cited on pages 14 and 34.)

[151] M. Ye, P. Yin, and W.-C. Lee. Location Recommendation for Location-based Social Networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 458–461. ACM, 2010. (Cited on page 1.)

[152] F. Yu, Q. Liu, S. Wu, L. Wang, and T. Tan. A Dynamic Recurrent Model for Next Basket Recommendation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 729–732. ACM, 2016. (Cited on pages 15 and 80.)

[153] T. Yu, Y. Shen, R. Zhang, X. Zeng, and H. Jin. Vision-Language Recommendation via Attribute Augmented Multimodal Reinforcement Learning. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 39–47. ACM, 2019. (Cited on page 79.)

[154] Z. Yuyan, S. Xiayao, and L. Yong. A Novel Movie Recommendation System based on Deep Reinforcement Learning with Prioritized Experience Replay. In *19th IEEE International Conference on Communication Technology*, pages 1496–1500. IEEE, 2019. (Cited on pages 77, 78, 79, and 85.)

[155] J. Zhang, B. Hao, B. Chen, C. Li, H. Chen, and J. Sun. Hierarchical Reinforcement Learning for Course Recommendation in MOOCs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 435–442. AAAI Press, 2019. (Cited on page 62.)

[156] S. Zhang and K. Balog. Evaluating Conversational Recommender Systems via User Simulation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1512–1520. ACM, 2020. (Cited on pages 56 and 80.)

[157] S. Zhang, L. Yao, A. Sun, and Y. Tay. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*, 52(1):5, 2019. (Cited on page 1.)

[158] Y. Zhang, F. Feng, X. He, T. Wei, C. Song, G. Ling, and Y. Zhang. Causal Intervention for Leveraging Popularity Bias in Recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–20. ACM, 2021. (Cited on pages 2, 16, 20, and 50.)

[159] C. Zhao and L. Hu. CapDRL: A Deep Capsule Reinforcement Learning for Movie Recommendation. In *PRICAI 2019: Trends in Artificial Intelligence: 16th Pacific Rim International Conference on Artificial Intelligence*, pages 734–739. Springer, 2019. (Cited on pages 77 and 79.)

[160] S. Zhao, I. King, and M. R. Lyu. A Survey of Point-of-Interest Recommendation in Location-based Social Networks. *arXiv preprint arXiv:1607.00647*, 2016. (Cited on page 1.)

[161] W. X. Zhao, S. Mu, Y. Hou, Z. Lin, Y. Chen, X. Pan, K. Li, Y. Lu, H. Wang, C. Tian, et al. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4653–4664. ACM, 2021. (Cited on page 15.)

[162] X. Zhao, L. Zhang, Z. Ding, D. Yin, Y. Zhao, and J. Tang. Deep Reinforcement Learning for List-wise Recommendations. *arXiv preprint arXiv:1801.00209*, 2017. (Cited on pages 61 and 79.)

[163] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang. Deep Reinforcement Learning for Page-wise Recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 95–103. ACM, 2018. (Cited on pages 56, 58, 61, 79, and 101.)

[164] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1040–1048. ACM, 2018. (Cited on pages 61, 78, 79, 80, 85, and 88.)

[165] X. Zhao, L. Xia, Z. Ding, D. Yin, and J. Tang. Toward Simulating Environments in Reinforcement Learning based Recommendations. *arXiv preprint arXiv:1906.11462*, 2019. (Cited on pages 5, 6, 56, 57, 62, 65, and 80.)

[166] X. Zhao, X. Zheng, X. Yang, X. Liu, and J. Tang. Jointly Learning to Recommend and Advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3319–3327. ACM, 2020. (Cited on pages 56, 57, 61, 78, 79, and 85.)

[167] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176. ACM, 2018. (Cited on pages 1, 56, 61, 77, 78, 79, 80, and 85.)

[168] Z. Zheng, Z. Qiu, T. Xu, X. Wu, X. Zhao, E. Chen, and H. Xiong. CBR: Context Bias Aware Recommendation for Debiasing User Modeling and Click Prediction. In *Proceedings of the ACM Web Conference 2022*, pages 2268–2276. ACM, 2022. (Cited on page 50.)

[169] H. Zhuang, Z. Qin, X. Wang, M. Bendersky, X. Qian, P. Hu, and D. C. Chen. Cross-Positional Attention for Debiasing Clicks. In *Proceedings of the Web Conference 2021*, pages 788–797. ACM / IW3C2, 2021. (Cited on pages 50 and 101.)

[170] L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, and D. Yin. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. *arXiv preprint arXiv:1902.05570*, 2019. (Cited on pages 1, 56, 58, and 68.)

Summary

Recommender systems have been widely deployed in various scenarios to help users quickly find what they need from a collection of items. They involve predicting user preferences for items and making recommendations based on the predicted user preferences. Predominant recommendation methods rely on supervised learning models to predict user ratings on items or the probabilities of users interacting with items. In addition, reinforcement learning models play a pivotal role in improving long-term user engagement within recommender systems. In practice, both of these recommendation methods are commonly trained on logged user interactions and, therefore, subject to bias present in logged user interactions. As a solution, the task of debiasing recommender systems has been proposed to mitigate the effect of bias on recommendation methods that learn from biased logged user interactions. This thesis concerns complex forms of bias in real-world user behaviors and aims to mitigate the effect of bias on reinforcement learning-based recommendation methods.

The first part of the thesis consists of two research chapters, each dedicated to tackling a specific form of bias: dynamic selection bias and multifactorial bias. In fact, real-world user behavior is better captured with these two complex forms of bias. To mitigate the effect of dynamic selection bias and multifactorial bias, we propose a bias propensity estimation method for each. By incorporating the results from the bias propensity estimation methods, the widely used inverse propensity scoring-based debiasing method can be extended to correct for the corresponding bias. Our experimental results show that the proposed debiasing methods outperform existing debiasing methods that assume static bias and single-factor bias.

The second part of the thesis consists of two chapters that concern the effect of bias on reinforcement learning-based recommendation methods. Its first chapter focuses on mitigating the effect of bias on simulators, which enables the learning and evaluation of reinforcement learning-based recommendation methods. Our experimental results confirm that bias present in logged data affects simulators and our proposed debiasing method can mitigate bias of simulators. Moreover, we make the debiased simulator publicly available to help researchers in the field develop and evaluate reinforcement learning-based recommendation methods. Its second chapter further explores different state encoders for reinforcement learning-based recommendation methods when learning and evaluating with the proposed debiased simulator.

In summary, we focus on different forms of bias in various recommendation methods, including both supervised learning and reinforcement learning-based methods. Throughout, my research goals have consistently revolved around comprehending the complexities of real-world user behavior, effectively mitigating the effect of bias on recommendation methods, and enhancing the effectiveness and robustness of debiased recommendation methods.

Samenvatting

Aanbevelingssystemen worden breed ingezet in verschillende scenario's om gebruikers snel te helpen vinden wat ze nodig hebben in een verzameling items. Ze behelzen het voorspellen van de voorkeuren van gebruikers voor items en het doen van aanbevelingen op basis van de voorspelde gebruikersvoorkeuren. Vooraanstaande aanbevelingsmethoden vertrouwen op *supervised* modellen om gebruikersbeoordelingen voor items of de waarschijnlijkheden van gebruikersinteracties met items te voorspellen. Daarnaast spelen modellen voor *reinforcement learning* een cruciale rol bij het verbeteren van de betrokkenheid van gebruikers op lange termijn binnen aanbevelingssystemen. Beide soorten aanbevelingsmethoden worden in de praktijk doorgaans getraind op gelogde gebruikersinteracties en zijn daarom onderhevig aan de aanwezige bias in deze gelogde gebruikersinteracties. Als oplossing is de taak van het debiasen van aanbevelingssystemen voorgesteld om het effect van bias op aanbevelingsmethoden die leren van bevooroordeelde, gelogde gebruikersinteracties te verminderen. Dit proefschrift richt zich op complexe vormen van bias in het gedrag van echte gebruikers en heeft tot doel het effect van bias op reinforcement learning-gebaseerde aanbevelingsmethoden te verminderen.

Het eerste deel van het proefschrift bestaat uit twee onderzoekshoofdstukken, elk gewijd aan het aanpakken van een specifieke vorm van bias: dynamische selectiebias en multifactoriale bias. In feite wordt het gedrag van echte gebruikers beter vastgelegd met behulp van deze twee complexe vormen van bias. Om het effect van dynamische selectiebias en multifactoriale bias te verminderen, stellen we een bias-voorkeurschattingmethode voor elk voor. Door de resultaten van de bias-voorkeurschattingmethoden op te nemen, kan de veelgebruikte debiasing-methode op basis van *inverse propensity*-scores worden uitgebreid om de overeenkomstige bias te corrigeren. Onze experimentele resultaten tonen aan dat de voorgestelde debiasing-methoden beter presteren dan bestaande debiasing-methoden die statische bias en single-factor bias veronderstellen.

Het tweede deel van het proefschrift bestaat uit twee hoofdstukken die betrekking hebben op het effect van bias op *reinforcement learning*-gebaseerde aanbevelingsmethoden. Het eerste hoofdstuk richt zich op het verminderen van het effect van bias op simulators, die het leren en evalueren van *reinforcement learning*-gebaseerde aanbevelingsmethoden mogelijk maken. Onze experimentele resultaten bevestigen dat bias aanwezig in geregistreerde gegevens van invloed is op simulatoren en onze voorgestelde debiasing-methode kan de bias van simulatoren verminderen. Bovendien stellen we de gedebiasde simulator openbaar beschikbaar om onderzoekers op dit gebied te helpen bij het ontwikkelen en evalueren van *reinforcement learning*-gebaseerde aanbevelingsmethoden. Het tweede hoofdstuk verkent verder verschillende state encoders voor *reinforcement learning*-gebaseerde aanbevelingsmethoden bij het leren en evalueren met de voorgestelde gedebiasde simulator.

Samengevat richten we ons op verschillende vormen van bias in verschillende aanbevelingsmethoden, inclusief zowel *supervised* als *reinforcement learning*-gebaseerde methoden. Gedurende het hele onderzoek waren mijn onderzoeksdoelen consequent gericht op het begrijpen van de complexiteit van het gedrag van echte gebruikers, het effectief verminderen van bias op aanbevelingsmethoden, en het verbeteren van de effectiviteit en robuustheid van gedebiasde aanbevelingsmethoden.