Personalized Recommendation Using Reinforcement Learning

Afsar, Mohammad Mehdi

http://hdl.handle.net/1880/114674

Downloaded from PRISM Repository, University of Calgary
UNIVERSITY OF CALGARY

Personalized Recommendation Using Reinforcement Learning

by

Mohammad Mehdi Afsar

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA
MAY, 2022

© Mohammad Mehdi Afsar 2022
Abstract

The massive volume of information available on the web leads to the problem of *information overload*, which makes it difficult for a decision maker to make right decisions. Recommender systems (RSs) are software tools and algorithms that have been developed with the idea of helping users find their items of interest through predicting their preferences or ratings on items. It has been shown that the problem of recommending items to the user could be considered as a sequential decision problem and be formulated as a Markov decision process, so reinforcement learning (RL) algorithms can be used to solve this problem. The primary aim of this dissertation is to investigate this topic and to propose new recommendation approaches using RL.

The first part of this thesis, namely chapters 2 and 3, presents a healthcare use case of intelligent agents and RSs. In particular, chapter 2 presents a high-level design, called ALAN, to play the role of a *patient decision aid* for prostate cancer patients. ALAN is a multi-layered, multi-agent system in which each agent is responsible to provide a specific service in order to facilitate shared decision making for these patients. Moreover, an article RS with learning ability is proposed in chapter 3 to represent the Learning agent in ALAN, which combines multi-armed bandits with knowledge-based RSs for the provision of information for cancer patients.

Motivated by the first part, the second part of this thesis (chapters 4 and 5) deeply explores the topic of recommendation using RL algorithms. More precisely, chapter 4 provides a thorough literature review on reinforcement learning based recommender systems (RLRSs). The main goal of this chapter is to provide a deep analysis of almost all important RLRSs proposed and to depict a vista toward the field since the beginning. This chapter illustrates the importance of deep RL (DRL) in reviving the use of RL for RSs. Chapter 5 is an outcome of this chapter, which tries to address an important problem of DRL when applied to real applications like RSs, i.e., *sample inefficiency*. In this chapter, RL is combined
with imitation learning in order to accelerate RL agent’s learning and to make it sample efficient.

Finally, chapter 6 proposes a new recommendation approach from a totally new perspective. This chapter borrows ideas from Computer Networks field, clustering in wireless sensor networks in particular, and presents a probabilistic recommendation approach that can balance the similarity-diversity trade-off. The proposed approach is simple, scalable, and completely explainable.
Preface

All chapters of this thesis, excluding Introduction and Conclusion chapters, have been published or submitted for publication as follows:


Other publications and presentations during my PhD studies:


Acknowledgements

First and foremost, I would like to thank Dr Behrouz Far, my supervisor, for his invaluable support during my PhD research journey. He is really one of the best supervisors one can ever have. He taught me to believe in myself and gave me the opportunity to research with absolute freedom. I always remember his sentence from one of our early meetings: “the graduate student and supervisor play in the same team, not in the opposite teams.” After this sentence, I found that I am in the right place to do my PhD studies. Dr. Far, thank you for agreeing to supervise me during this challenging and exciting journey! I am also grateful to Dr Trafford Crump, my co-supervisor, for his financial support and continuous guidance during these four years. Dr Crump, this research was impossible without your invaluable support. Thank you!

Many thanks to Dr Kartikeya Murari, Dr Svetlana Yanushkevich, and Dr Marek Reformat, the other members of my defence committee, for reviewing this thesis and your constructional feedbacks. I also appreciate the financial support provided by Eyes High and Alberta Graduate Excellence Scholarships.

Special thanks to Dr Mohammad Tayarani who was the first person taught me how to conduct quality research. Your patience and passion for research always inspired me to do my best in my research. My sincere gratitude also goes to Dr Mohamed Younis, someone with whom I had the opportunity to collaborate and has always been a major source of support. Dr Tayarani and Dr Younis, it was impossible to reach this point in my life without your continuous support. Thank you!
I am extremely grateful to my mom, Esmat, and my dad, Aliasghar, for your support, love, and encouragement. Without your tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my PhD studies. I dedicate this thesis to my parents.
# Table of Contents

Abstract ii  
Preface iv  
Acknowledgements vi  
Table of Contents viii  
List of Figures and Illustrations xi  
List of Tables xiii  

1 Introduction  1  
1.1 Thesis Motivation and Background . . . . . . . . . . . . . . . . . . . . . . . 2  
1.2 Thesis Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4  
1.3 Thesis Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5  
1.4 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7  

2 Intelligent Multi-Purpose Healthcare Bot Facilitating Shared Decision Making  9  
2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10  
2.2 The Proposed System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11  
2.3 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14  

3 An Exploration On-demand Article Recommender System for Cancer Patients Information Provisioning  15  
3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16  
3.2 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18  
3.2.1 Knowledge-based Recommender Systems (KBRSs) . . . . . . . . . . . 18  
3.2.2 Bandits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19  
3.3 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20  
3.4 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20  
3.4.1 Rationale . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.2</td>
<td>Proposed KERS</td>
<td>22</td>
</tr>
<tr>
<td>3.5</td>
<td>Experiments</td>
<td>24</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Setup</td>
<td>24</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Results</td>
<td>26</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Discussion</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion and Future Work</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Reinforcement Learning based Recommender Systems: A Survey</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Preliminaries</td>
<td>37</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Recommender Systems</td>
<td>37</td>
</tr>
<tr>
<td>4.2.2</td>
<td>From Reinforcement Learning to Deep Reinforcement Learning</td>
<td>38</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Why Reinforcement Learning for Recommendation?</td>
<td>47</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Problem Formulation</td>
<td>47</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Proposed RLRS Framework</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>Reinforcement Learning based Recommender Systems Algorithms</td>
<td>53</td>
</tr>
<tr>
<td>4.3.1</td>
<td>RL-based RSs</td>
<td>53</td>
</tr>
<tr>
<td>4.3.2</td>
<td>DRL-based RSs</td>
<td>59</td>
</tr>
<tr>
<td>4.4</td>
<td>Emerging Topics</td>
<td>77</td>
</tr>
<tr>
<td>4.5</td>
<td>Open Research Directions</td>
<td>83</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>86</td>
</tr>
<tr>
<td>5</td>
<td>Sample Efficiency in Deep Reinforcement Learning based Recommender Systems with Imitation Learning</td>
<td>87</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>88</td>
</tr>
<tr>
<td>5.2</td>
<td>Related Work</td>
<td>90</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Collaborative Filtering</td>
<td>90</td>
</tr>
<tr>
<td>5.2.2</td>
<td>RLRSs and Sample Inefficiency</td>
<td>90</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Combining RL with IL</td>
<td>91</td>
</tr>
<tr>
<td>5.3</td>
<td>Methodology</td>
<td>92</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Problem Formulation</td>
<td>92</td>
</tr>
<tr>
<td>5.3.2</td>
<td>DQN</td>
<td>93</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Neighborhood CF</td>
<td>93</td>
</tr>
<tr>
<td>5.3.4</td>
<td>IR2S</td>
<td>94</td>
</tr>
<tr>
<td>5.4</td>
<td>Experiments</td>
<td>95</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Setup</td>
<td>95</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Results</td>
<td>97</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusion and Future Work</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>Balancing Similarity-Diversity Trade-off in Recommender Systems: a Probabilistic Approach</td>
<td>102</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>103</td>
</tr>
<tr>
<td>6.2</td>
<td>Background</td>
<td>106</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Recommender Systems</td>
<td>106</td>
</tr>
<tr>
<td>6.2.2</td>
<td>LEACH</td>
<td>107</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.3</td>
<td>Methodology</td>
<td>108</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Problem Definition</td>
<td>108</td>
</tr>
<tr>
<td>6.3.2</td>
<td>The Proposed PRS</td>
<td>109</td>
</tr>
<tr>
<td>6.4</td>
<td>Experiments</td>
<td>112</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Setup</td>
<td>113</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Results</td>
<td>117</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Discussion and Future Plan</td>
<td>124</td>
</tr>
<tr>
<td>6.5</td>
<td>Conclusion</td>
<td>125</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Future Work</td>
<td>127</td>
</tr>
<tr>
<td>7.1</td>
<td>Thesis Summary</td>
<td>127</td>
</tr>
<tr>
<td>7.2</td>
<td>Future Work</td>
<td>130</td>
</tr>
</tbody>
</table>

Bibliography | 132 |
## List of Figures and Illustrations

1.1 Thesis outline .......................................................... 8

2.1 The multi-layered architecture of ALAN ................................. 13

3.1 The proposed system architecture ........................................ 24
3.2 The cumulative reward for the three algorithms on prostate cancer dataset ................................. 27
3.3 The cumulative reward for the three algorithms on BBC dataset ................................. 28
3.4 The number of times each arm played by algorithms (\(M = 1\) and \(\eta = 1\)) .................. 29

4.1 Publications information of 97 surveyed RLRS papers. (a) Distribution of RLRSs publications per year (until September 2021) separated based on RL and DRL methods. (b) Venue distribution of published RLRSs. To be clear, we filtered venues with only one publication and termed them as ‘Others’ in the graph. This includes a long list of venues, including AAMAS, ICML, ICDM, and JMLR. (c) The proportion of different types of publications, i.e., conference proceedings, journal articles, and arXiv preprints, of surveyed RLRSs. 34
4.2 Agent-environment RL interface ........................................ 39
4.3 RL algorithms used by RLRSs ........................................... 40
4.4 The proposed RLRS framework .......................................... 49
4.5 (a) SR1, (b) SR2, (c) SR3. The dashed Encoding module in SR3 means that some models merely use input embeddings as states. ................... 50
4.6 The concept of sliding window used in [1] for state representation .............. 54
4.7 The summary of four components of RLRS framework in RL-based methods ................................. 55
4.8 The summary of four components of RLRS framework in DRL-based methods ................................. 61
4.9 The architecture of DEERS [2] with two input states ................................. 62
4.10 (a) Q network A1 architecture, (b) Q network A2 architecture ................................. 62
4.11 State representation module in DRR [3, 4] ................................ 63
4.12 Dueling architecture used in DRN [5] ................................... 66
4.13 The neural architecture of policy \(\pi_{\theta} [6]\) .................................. 69
4.15 (a) MADDPG architecture, (b) MASSA architecture ................................. 73
4.16 Graph-based reasoning example [8] ..................................... 81
4.17 Evaluation metrics and datasets used more often by RLRSs ................................. 85
5.1 Parameter Study. The effect of different values of $I$ (similar items) on HR and NDCG (ML100K) ........................................ 98
5.2 Parameter Study. The effect of different values of $I$ (similar items) on HR and NDCG (ML1M) ........................................ 98
5.3 Parameter Study. The effect of different values of $U$ and $I$ (similar users and items) on HR and NDCG (ML100K). Vertical labels on x-axis indicate $I$ values. 99
5.4 Parameter Study. The effect of different values of $U$ and $I$ (similar users and items) on HR and NDCG (ML1M). Vertical labels on x-axis indicate $I$ values. 99
5.5 Comparing performance between algorithms (ML100K) ....................... 100
5.6 Comparing performance between algorithms (ML1M) ....................... 101

6.1 The effect of $p$ on $\mathcal{R}$ in B-PRS ........................................ 118
6.2 The effect of $p$ on $\mathcal{D}$ in B-PRS ........................................ 119
6.3 The effect of $p$ on $\mathcal{R}$ in P-PRS ........................................ 120
6.4 The effect of $p$ on $\mathcal{D}$ in P-PRS ........................................ 120
6.5 The effect of $\lambda$ and $Q_{th}$ on $\mathcal{R}$ in H-PRS .......................... 121
6.6 The effect of $\lambda$ and $Q_{th}$ on $\mathcal{D}$ in H-PRS .......................... 122
6.7 Comparison between algorithms and the effect of $k$ on their performance .... 123
6.8 Comparison between algorithms and the effect of $M$ on their performance .... 123
6.9 Comparing diversity between algorithms ................................. 124
## List of Tables

3.1 Notation Description .................................................. 21
4.1 Abbreviation Definition ................................................. 52
4.2 RL-based Methods ....................................................... 53
4.3 DRL-based Methods ...................................................... 60
4.4 DQN-based RSs ........................................................... 65
4.5 DDPG-based RSs .......................................................... 72
5.1 MovieLens Datasets Statistics .......................................... 96
5.2 Hyper-parameters Values .............................................. 97
6.1 Parameters Values ....................................................... 115
6.2 DQN Hyper-parameters Values ....................................... 116
Chapter 1

Introduction

Recommender systems (RSs) help millions of users on a daily basis to find their items of interest. They are the core technology that recommends personalized videos on YouTube, applications on Google Play Store, movies on Netflix, and products on Amazon. Providing personalized recommendations is not a trivial task and it is increasingly becoming more complex due to the explosion of the data on the web and the huge number of Internet users. This makes traditional recommendation approaches ineffective and emphasizes the importance of devising new recommendation algorithms and techniques. With recent advances in the machine learning field, including deep learning, reinforcement learning (RL), and deep reinforcement learning (DRL), this thesis proposes and develops new recommendation algorithms to address the problems of traditional recommendation methods discussed in the following sections.

This chapter is organized as follows. Section 1.1 provides a brief background and thesis motivation. In section 1.2, the objectives targeted by this thesis are discussed. The main contributions of this dissertation are explained in section 1.3 and finally the thesis outline is given in section 1.4.
1.1 Thesis Motivation and Background

RSs have become an inseparable part of our everyday lives. They help us find our favorite items to purchase, our friends on social networks, and our favorite movies to watch. Today, RSs are an essential part of most giant companies, like Google, Facebook, Amazon, and Netflix, and employed in a wide range of applications, including entertainment [9, 10, 11], e-commerce [12], news [13], e-learning [14], and healthcare [15]. The problems of traditional recommendation methods, i.e., collaborative filtering and content-based filtering, include cold-start, lack of diversity, scalability, and low quality recommendation [16]. Recently, there has been a significant body of research on deep learning based RSs [17]. Nonetheless, these methods typically consider the recommendation problem as a static prediction problem and fail to take into account the sequential, dynamic, and long-term user engagement with the system, something that reinforcement learning (RL) is able to handle well. It has been shown that the recommendation problem can be formulated as a sequential decision making problem in the form of a Markov decision process (MDP) and be solved by RL algorithms [18, 19]. Hence, this thesis pursues new approaches, primarily using RL, to alleviate part, if not all, of these problems and to provide better and more personalized recommendations.

The primary idea in the first part (chapters 2 and 3) is to develop a new generation of patient decision aids for prostate cancer patients based on intelligent agents and RSs. Prostate cancer is the most common cancer among men in Canada. In 2021, there was an estimated number of 24,000 prostate cancer diagnoses and 4,500 deaths because of this disease\(^1\). Patient decision aids have been developed to facilitate shared-decision making, and consequently, to improve the quality of life of these patients. Current patient decision aids are typically in the form of booklet, DVD, or website. However, these tools are usually impractical, not personalized, unreliable, and incomprehensible [20, 21, 22, 23, 24]. Due to the importance of the topic and inapplicability of current patent decision aids, we decided

\(^1\)Canadian Cancer Society
to design a next generation of patent decision aids using intelligent agent technology. After carefully researching relevant topics, I designed ALAN: a multi-layered, multi-agent system (details in chapter 2). ALAN is able to deliver high quality care and to address design features necessary to facilitate support and shared-decision making.

ALAN is a general framework composed of various agents to fulfil various requirements, set by healthcare providers, and each researcher in our team was responsible to develop a specific agent. My task was to develop the Learning agent in the Analytical Decision-Making layer. The idea behind the Learning agent was to know users (patients and clinicians), to learn their preferences, and to evolve ALAN’s behavior according to changes in user responses. The result of my investigation was that an RS could play an effective role in learning users preferences and educating them by providing personalized recommendations tailored to their situation and information needs. Chapter 3 is the result of my studies to develop an article RS with learning ability, which can provide useful information for cancer patients. I have developed a new article RS based on multi-armed bandits (MABs) and knowledge-based systems.

MABs has been developed to alleviate the classical problem of exploration vs. exploitation. In MABs, the agent should maximize a numerical reward by selecting the best arm among $k$ arms [25]. While MABs provides a useful and simple framework to enable the agent with learning ability, it is limited in modelling the environment, mainly due to the fact that it is stateless. That means, the state of the environment in MABs is assumed to be constant during the interaction between the agent and the environment. This motivated me to explore the full RL and its effect on the performance of RSs. The result of this investigation is presented in the second part (chapters 4 and 5) of the thesis. More precisely, I started to deeply study reinforcement learning based recommender systems (RLRSs) and came up with the idea of preparing a comprehensive survey on RLRSs (chapter 4).

This survey provided me with a deep knowledge on RLRSs and helped find out what gaps exist in the field. For example, I found that there is still a serious problem in applying DRL
to real world applications due to the sample inefficiency problem, which means the DRL agent needs a long time to interact with the environment and to garner useful experiences. Although this might be acceptable in simulations and games, it is definitely not tolerated in a sensitive field like RSs where the user of the system is a human and they quickly become bored and leave the system with irrelevant recommendations. In chapter 5, I target this problem by proposing a method to combine RL with imitation learning.

Finally, I found that although RL is an effective method to provide better recommendations, there are still some inherent challenges in RL, like lack of explainability, slow or unstable convergence, and difficult reward formulation. I was looking for designing a new recommendation approach that not only provides accurate recommendations, but it can also provide diverse recommendations. I decided to leverage my knowledge in the Computer Networks field and borrow ideas from that field to propose a new recommendation approach. Hence, chapter 6 provides an interesting combination between RS and Computer Networks fields. In particular, I propose a new probabilistic recommendation approach based on ideas from clustering in sensor networks, which is able to balance the similarity-diversity trade-off and to provide explainable recommendations.

1.2 Thesis Objectives

The overarching goal of this thesis is to develop new recommendation algorithms to provide better and more personalized recommendations. The detailed objectives of the thesis regarding individual chapters are as follows:

- Chapter 2 aims at providing a next generation patient decision aid for prostate cancer patients by using intelligent agent technology.

- The objective of chapter 3 is to develop an article RS with learning ability to provide useful and personalized information for cancer patients.
• Chapter 4 is a deep exploration of the RLRS field, whose objective is to deeply study existing RLRSs since the beginning, providing a comprehensive survey on the state-of-the-art, and to gain a good understanding of key concepts in the field.

• Chapter 5 pursues the objective of addressing the sample inefficiency problem in RLRSs.

• The primary objective in chapter 6 is to propose an explainable, probabilistic recommendation approach that balances the similarity-diversity trade-off in RSs.

1.3 Thesis Contributions

The contribution of each chapter is as follows:

Chapter 2 Intelligent Multi-Purpose Healthcare Bot Facilitating Shared Decision Making

The first contribution of this thesis is the design of a multi-layered multi-agent system, ALAN, that serves as a patient decision aid to facilitate shared-decision making process. This design is a blueprint for creating agents, providing a clear map to show the role of each agent and relation between them. This chapter also discusses about some practical contributions and proof of concepts developed. For example, we have developed an interactive system that can understand the conversation between a patient and physician during the patient visit and fetches useful information from database/cloud and presents it to them in a logical way (e.g., graphs, tables, etc.).

Chapter 3 An Exploration On-demand Article Recommender System for Cancer Patients Information Provisioning

The second contribution of this thesis is to design and implement an article RS, called KERS, that provides informative recommendations for cancer patients; a possible design for the Learning agent in ALAN. Because of its healthcare use case, KERS uses a
knowledge-base to provide reliable recommendations, which is developed by a domain expert. To enable KERS with learning ability, MABs is used. The main theoretical contribution of this chapter is to develop a new hybrid system of MABs and knowledge-based RSs. This consists of introducing a novel on-demand exploration method to quickly find users interests and utilize them in the next steps. Moreover, in order to evaluate the performance of the proposed KERS, a flexible simulator is developed that provides different options to model a user.

Chapter 4 Reinforcement Learning based Recommender Systems: A Survey
The next contribution of this dissertation is to present the first and most comprehensive survey on RLRSs. First it is recognized that the research in the RLRS field could be divided into RL and DRL-based methods. Then, in order to consolidate the advancement in the field, an RLRS framework is proposed with four components: state representation, policy optimization, reward formulation, and environment building, which can be used to formulate any RLRS. In order to help those who are not familiar with RL, this chapter also provides a thorough background on RL algorithms used by RLRSs. Besides, important trends in each component of the the RLRS framework and emerging topics in the RLRS field in general are highlighted. Finally, to advance the filed in the future, some open research directions are discussed.

Chapter 5 Sample Efficiency in Deep Reinforcement Learning based Recommender Systems with Imitation Learning
This chapter proposes a novel hybrid method to tackle the sample inefficiency problem in RLRSs. In particular, the proposed method, called IR2S, combines RL with imitation learning. It assumes that demonstrations are available in the form of user ratings and provides a flexible framework to leverage these demonstrations to accelerate DRL agent’s learning. IR2S utilizes demonstrations using neighborhood collaborative filtering techniques, namely item-based and user-based collaborative filtering. The per-
formance of IR2S is evaluated in an offline study on two popular datasets, MovieLens 100K and 1M, and the results of experiments confirm the effectiveness of the proposed IR2S.

Chapter 6 Balancing Similarity-Diversity Trade-off in Recommender Systems: a Probabilistic Approach

A new probabilistic recommendation approach, called PRS, is proposed in this chapter. PRS borrows ideas from cluster head selection in wireless sensor networks and provides a framework that is able to balance the similarity-diversity trade-off in RSs. I propose three versions for PRS: Basic PRS (B-PRS), Priority PRS (P-PRS), and Heterogeneous PRS (H-PRS). In B-PRS, there is no difference between items and all items have the same chance to be recommended to the user. In P-PRS, items with higher priority (importance) are recommended to the user more often. Finally, H-PRS considers a heterogeneity between items and provides a flexible framework to balance the similarity and diversity trade-off. To effectively measure diversity, this chapter proposes a new diversity metric that emphasizes the importance of diversity between consecutive lists of items. The performance of the proposed PRS is evaluated using a simulation study conducted on RecSim [26], a simulator developed by Google.

1.4 Thesis Outline

This dissertation composes of an introduction chapter, five main chapters, and a conclusion and future work chapter. Fig. 1.1 illustrates the thesis outline. First part (chapters 2 and 3) shows how the agent technology and RSs can be utilized to serve as a patient decision aid and to facilitate shared-decision making process. As depicted in this figure, chapter 3 motivates the research in the second part, exploring RLRSs and analyzing how full RL can be used for RSs. The result of this investigation is presented in chapters 4 and 5, where I deeply study RLRSs and propose a solution to address the sample inefficiency problem in RLRSs. The
deep analysis performed in chapter 4 helps understand that even RL is not able to address all problems in RSs. Therefore, in chapter 6, two RS and Computer Networks fields are merged in the form of a probabilistic recommendation approach, which is able to alleviate many RS problems. Finally, in chapter 7, the thesis is concluded and some open directions for the future work are suggested.
Intelligent Multi-Purpose Healthcare Bot
Facilitating Shared Decision Making

Abstract

Patient decision aids (PtDAs) have been promoted to facilitate personalized information retrieval and decision support; nonetheless, although promoted for more than 20 years, they have generally failed to gain a foothold in the general delivery of healthcare. Intelligent interactive agent technologies could address the design features necessary to facilitate support and shared-decision making. In this thesis, we develop and build a PtDA for prostate cancer using intelligent agent technology. The proposed system, called ALAN, has a multi-layered architecture with three layers. While the first layer (User-Interface) is responsible to effectively interact with users (patients and physicians), the bottom layer (Data) handles requests regarding storing and retrieving the data. Unlike most existing bots, our core objective is to enable ALAN with learning abilities, which can evolve in the course of time and improve its behaviour with minimum distraction of the user. To this end, reinforcement learning and deep learning algorithms are employed in the main layer, i.e., Analytical Decision Making. This research is expected to have impact on delivery of personalized healthcare.
2.1 Introduction

There are many decisions in healthcare where the scientific evidence supporting the best choice is insufficient or equivocal. These decisions are preference-sensitive, because the ultimate choice should depend on patients’ preferences for the benefits, harms, and scientific uncertainties associated with the medically reasonable alternatives [27]. For preference-sensitive decisions, shared-decision making is advocated as a way of engaging and informing patients to make a value-based choice and delivering high quality care [28].

Patient decision aids (PtDAs) have been promoted to facilitate shared-decision making, and there has been considerable scientific study into the use of these instruments for this purpose [29]. PtDAs provide patients with balanced information about their options, while eliciting their preferences and any areas of cognitive dissonance. Nonetheless, despite being introduced over 20 years ago, PtDAs have generally failed to gain a foothold in the delivery of healthcare. Common PtDAs barriers, such as a lack of applicability due to patient characteristics and/or a specific clinical situation [29], are further compounded by healthcare providers who are overworked [30], time-constrained, and/or improperly trained on how to administer and use PtDAs in their clinical practice [31].

Recent advances in intelligent agent technology (i.e., “chatbots”) could address the design features necessary to support a “next generation” of PtDAs, and thus facilitate improved shared-decision making using agent-based systems. Agents are software entities that can interact with each other and human counterparts to solve problems; to share expertise; to work in parallel or sequence on common problems; to represent multiple viewpoints and the knowledge of multiple experts; and to compete for limited resources.

In this thesis, we utilize the intelligent agent technology to facilitate and enhance shared-decision making in healthcare. We propose ALAN, an Artificial intelligence- and Learning-based multi-Agent system for Next-generation PtDAs. ALAN is composed of three layers, namely the User-Interface (UI), Analytical Decision-Making (ADM), and Data layers. The
UI has an interactive interface to communicate with users in a natural way, and its main task is to receive the request through natural language and show the response in the most logical form. On the other hand, the ADM is behind the scene and responsible for learning, making important decisions, and analysis. In the bottom, all the data is stored and retrieved in the data layer. Unlike existing virtual assistants and chatbots that are mostly deterministic [32], the main feature of ALAN is its ability in learning and making smart decisions. In this regard, the Learning agent, located in the ADM, plays a critical role and is equipped with Reinforcement Learning (RL) algorithms.

2.2 The Proposed System

As mentioned earlier, ALAN has a multi-layered architecture with three layers. We have devised incremental development and prototyping approach for ALAN. The first prototype includes the development of the UI and Data layers. The UI (the top layer) is responsible to effectively interact with users (patients and physicians) and is composed of three agents, i.e., Human Interaction (HI), Personal Assistant (PA), and Survey agents. The main duties of the HI can be summarized as receiving requests from the users, deciding to query which agent based on the request of users, and showing the returned answer from other agents in an appropriate way. The PA, on the other hand, has four main tasks, namely handling referrals, booking appointments, billing services, and external communications. Also, the Survey agent is in charge of gathering data from patients and uploading this data to the system. As a proof of concept, we have already developed a voice-activated interactive prototype system that understands the conversation between a patient and a physician during the patient’s visit session and fetches information from a private cloud setup and presents it to them without interfering with the conversation. Note that we have used IBM Watson [33] for speech recognition. Moreover, we have developed a web-based application, called PROCheck, to have an interactive environment with patients. The Survey agent pushes surveys to patients
through PROCheck and the filled out surveys are saved to the database again using the Survey agent.

On the other hand, the bottom layer is the Data layer, which is composed of Database agent and Ontology agent. While the Database agent is in charge of handling common database operations, including data retrieval, storing, deletion, and search, the Ontology agent is responsible to modify the ontology tree, which keeps high-level and conceptual information in an hierarchical manner and also information for mapping different data in the database. We already have a mechanism in place to collect patient information using instruments, such as OAB-V8 and the Expanded Prostate Index Composite-26 (EPIC-26) domain scores. We leverage this and other data sources (provided by the Prostate Cancer Centre and Community Health Sciences Department, etc.) for intelligent data aggregation, predictive analytics, and personalized patient care. Not only our database contains patient information, but it also keeps other types of data, including the opinion of patients about their disease collected from online discussion forums, description of medical terms and diseases, training data for our Learning agent, and different documents. We have already developed a mechanism to crawl online discussion forums in order to collect the opinion of patients about their diseases. This data is particularly useful when used by Natural Language Processing (NLP) techniques to figure out relationships between patient pre- and post-treatment conditions and to find patterns for improved healthcare.

The second prototype includes the development of the ADM layer (the middle layer), which is composed of three agents, namely Learning, Decision Making (DM), and Analytics agents. The DM agent is queried by other agents when they need to come up with a decision about an important case. Machine learning and data fusion techniques are used in this agent to make the decision. A good example to illustrate the importance of the Decision-making agent is our previous work, LOWA [34], which was used in classifying breast tumor classification. The Analytics agent is mainly responsible for data analysis. Different machine learning algorithms and prediction models are used to analyze the data. For instance, when
A referral letter is received and processed by the PA, the Analytics agent is used to predict the disease and risk levels based on disease parameters extracted from the referral letter received. Fig. 2.1 depicts the multi-layered architecture of ALAN.

Our major objective and novel research part of this study is to enable ALAN to preserve the conversation in the most natural way possible and to improve its behaviour (answers or services) during the time and with minimum distraction of the user. To this end, we will employ RL and deep learning algorithms for the Learning agent. To explain the importance and procedure of the Learning agent, consider the following scenario. ALAN is queried by a physician to provide the possible treatment for prostate cancer patients. The system explores its repository, pulls the treatment types and shows the list. The very first option for this scenario is to use NLP techniques, such as lexical matching and semantic search. These techniques are helpful for a very small repository where the similarity between documents is very low; however, when we started to grow our repository and tested different queries, the accuracy of these techniques considerably diminished. More importantly, our target is
to have an interactive search based on the opinion of the physician where there is no need to a rich dataset to train our model. This is mainly because the number of training dataset for this purpose is scarce [35]. Accordingly, we utilize RL algorithms.

We model our information retrieval problem as a Markov Decision Process (MDP) where it learns to optimize the search using local repository and the Internet search, if required. To represent the MDP, we use a tuple $< S, A, R, T >$, where $S$ is the set of all states, $A$ is the set of all actions, $R$ is the reward function, and $T$ is the transition function. The state is the confidence of our algorithm in finding the similar documents. The action is the decision of what to do next according to the current state. The agent can choose to do a new search or to return a set of documents it thinks they are relevant. The main strategy in selecting rewards is to maximize the accuracy of search while minimizing the number of feedback the agent needs from the user. To solve this problem, we plan to try different RL algorithms, including Q-Learning and PPO (proximal policy optimization) [36].

### 2.3 Conclusion

We propose ALAN, a multi-purpose healthcare bot for a next-generation PtDA. ALAN utilizes both multi-layer architecture and multi-agent system technologies to provide a broad range of medical services. The main feature of ALAN is its ability in learning and improving its behaviour and services in the course of time. In the future, our focus is mainly on the Learning agent and employing RL algorithms.
Chapter 3

An Exploration On-demand Article Recommender System for Cancer Patients Information Provisioning

Abstract

Information provision plays an important role in educating patients with serious illnesses, like cancer, to cope with their disease conditions and to actively participate in shared-decision making process. Recent studies suggest that there is a lack of appropriate educational resources for such patients, specifically prostate cancer patients. To address this issue, in this paper, a Knowledge-based Exploration on-demand article Recommender System (called KERS) is proposed that can provide evidence-based information for patients. Recognizing the fact that exploration is expensive when the user of the system is a human, the main idea in KERS is to minimize exploration while achieving the maximum long-term satisfaction. Therefore, using a knowledge-base developed by an expert in the field, KERS learns user interests as quickly as possible and then it exploits this knowledge to recommend the best articles. Furthermore, KERS needs no information from users beforehand and it learns them through interacting with users. The system will help patients make informed decisions,
and at the same time, will reduce the burden on the healthcare providers. The results of experiments have confirmed the effectiveness of the proposed system compared to baseline methods.

3.1 Introduction

Prostate cancer is the most commonly diagnosed cancer among Canadian men; one in nine men in Canada are diagnosed with prostate cancer in their lifetime\(^1\). In 2020, there will be an estimated 23,300 Canadian men diagnosed with prostate cancer and 4,200 deaths due to this disease\(^2\). Coping with prostate cancer can be particularly difficult because the optimal treatment is not clear and treating the disease can have consequences for a man’s quality of life \([37, 38]\), including urinary incontinence, sexual dysfunction, depression, and anxiety \([24]\). Accordingly, providing prostate cancer patients with appropriate educational resources can play an important role in treatment decision making. Nonetheless, there is a deficiency in the availability of high-quality educational resources for prostate cancer patients compared to other cancer patients \([39, 24]\).

In cancer care, the provision of information has four purposes, to: 1) prepare patients for their treatment, 2) increase adherence to therapy, 3) increase their abilities to cope with the illness, and 4) to promote recovery \([40]\). Useful information is defined by providing patients with the kind of information they need to know, at the time in their disease progression that they need to know it, by the right source \([41]\). The benefits of information for cancer patients include increased satisfaction and involvement in decision making, improved ability in coping with stressful stages of disease, anxiety control, and enhanced communication with family members and clinicians \([41]\).

There is evidence that the satisfaction of prostate cancer patients has been improved with simple self-management educational resources, in the form of booklets \([24]\). However,\(^1\)Prostate Cancer Canada\(^2\)Canadian Cancer Society
many of these resources have been found to be too simple/impractical [20], unreliable [21, 22], incomprehensible [23], and not personalized [24]. Therefore, it seems with more sophisticated educational resources, there is a chance to better improve the satisfaction, and perhaps the quality of life, of these patients. We hypothesize that a recommender system (RS) could be designed to provide prostate cancer patients with useful information as a basis for improving their educational resources.

RSs are software tools that help users find items of their interest. There are two main paradigms toward RSs: collaborative filtering (CF) and content-based filtering (CBF). In CF, which is the foundation of initial RSs [16], the idea is to first find users similar to the main user, then recommend items liked by these similar users to the user. In CBF, items similar to the history of the user are recommended to the user. However, these methods are not effective when there is no database with enough user ratings. Knowledge-base RSs (KBRSs) are a third paradigm in which recommendations are generated using a knowledge-base (KB) developed by an expert in the domain. If appropriately developed, KBRSs can cover the problems of the two aforementioned methods, including new user/item (cold start), unreliable recommendations, and serendipity [42].

In this paper, we aim at tackling the information provisioning problem for prostate cancer patients through proposing a Knowledge-based, Exploration on-demand article Recommender System, called KERS. Because exploration is costly when recommending to human users, the main idea in KERS is to minimize the exploration while achieving the maximum satisfaction. To do this, we incorporate a new exploration vs. exploitation strategy into the KBRS technology. More specifically, KERS is composed of two phases: Exploration and Exploitation. User interests are learned through Exploration phase with the help of a KB. Then, this knowledge is used to recommend the best articles to the user in the Exploitation phase. The performance of KERS is validated through extensive experiments. In general, the contribution of this paper is threefold: 1) KERS is the first RS designed to cover the information needs of prostate cancer patients, 2) To our knowledge, this hybrid of KB and
learning ability (balancing exploration vs. exploitation) is new, and 3) We present a simple, flexible user simulator to evaluate our method offline through a simulation study.

The remaining of this paper is organized as follows. Section 3.2 provides necessary background for this paper. Section 3.3 defines the problem and explains notations used. The proposed KERS is described in Section 3.4. The results of experiments are presented in Section 3.5 and the paper is concluded in Section 3.6.

3.2 Background

Since both KBRSs and bandits inspired us in developing KERS, we provide a quick background for them below.

3.2.1 Knowledge-based Recommender Systems (KBRSs)

KBRSs use the knowledge provided by a human expert to generate recommendations. Instead of using user ratings to figure out the taste of the user, they rely on deep knowledge about a topic and explicit user requirements to come up with the best recommendations [43]. Thus, this type of RSs is a good match for applications where user ratings on items is scarce. Methods developed for KBRSs can be generally divided into case-based and constraint-based approaches [43]. The idea in both methods is the same; the user poses his problem or user requirements are collected, repairs are handled when there are discrepancies, and recommendations are made and explained [42, 43]. The difference is in the way how these recommendations are generated; while case-based KBRSs use a similarity metric to match user’s requirements with items in the KB [44, 45, 46, 47, 48], constraint-based methods rely on pre-defined and strong rules to match these two [49, 50, 43].
3.2.2 Bandits

The problem of balancing the exploration vs. exploitation is typically formulated as a multi-armed bandit (MAB) problem. In a MAB problem, the agent faces with a choice among \( k \) options and through selecting one choice, it receives a numerical reward [51]. This scenario is similar to a situation where a gambler should decide to play which lever of a slot machine to gain more payoffs in a series of trials. Algorithms developed to solve the MAB problem can be generally divided into context-free and contextual bandit algorithms [51, 52]. In the former, the agent knows nothing about the environment (items and users). On the other hand, in contextual bandits, the agent sees a feature vector of the environment alongside the history of each arm in order to select the best arm. Since we are inspired by context-free bandits in developing KERS, we provide more details about this type of bandits. \( \epsilon \)-greedy is one of the simplest context-free bandits in which either the best arm with probability \( 1 - \epsilon \) or a random arm with probability \( \epsilon \) is selected [51]. To address the unguided exploration by \( \epsilon \)-greedy, upper confidence bound (UCB) [53] algorithms follow optimism in the face of uncertainty. More specifically, these algorithms usually compute a confidence bound for each arm (action) and then select the arm with the largest bound. In other words, the less confident they are about an arm, the more likely they select it. From this family, UCB1 algorithm first plays each arm once. Then, at time step \( t \), it selects the arm that maximizes 
\[ Q_t(a) + \sqrt{\frac{2 \log t}{n_a}} \]
where \( Q_t(a) \) is the value of arm \( a \) at time step \( t \) and \( n_a \) is the number of times that arm \( a \) has been played. We compare the performance of KERS with \( \epsilon \)-greedy and UCB1 in Section 3.5.

In contrast, KERS is a hybrid method of KBRSs and MABs. On the one hand, it relies on the strength of KBs in terms of carefully classified information. On the other hand, it wisely modifies the unguided exploration in MABs and equips the KBRS with a new learning algorithm.
3.3 Problem Definition

We assume that KERS runs in a discrete time space, called *time steps*, \( t = 1, 2, 3, \ldots T \), where \( T \) is a finite number. There are \( M \) users and \( N \) items (articles in our case) in the system and the problem is to recommend, at time step \( t \), the best item to the active user. In every time step, the user is provided with \( k \) articles and the system receives the feedback from the user. Because articles in our KB (\( \mathcal{K} \)) are carefully developed and every one of them is about a specific topic, it is assumed that the title of articles is informative enough to convey the intent of every article. Thus, to not overwhelm the users, the system only shows the title of articles, which is no more than few words. On receiving every recommendation, the user provides a numerical *reward* \( (r) \). In this paper, \( r \) is considered to be the click of the user on the title of an article, which is equivalent to a numerical reward of 1 if clicked and 0 otherwise. That said, denoting \( a \) as an arm and \( A \) as the set of arms, the objective is

\[
\max R_A(T), \tag{3.1}
\]

where \( R_A(T) = \sum_{t=1}^{T} \sum_{i=1}^{k} r_{t,a_i} \). That means, the objective at \( t \) is to recommend the best \( k \) arms that maximize the cumulative reward during \( t = 1, \ldots, T \). In order to be consistent with bandit field’s terminology, *arm* and *article* terms are used interchangeably in this paper. Table 3.1 demonstrates the notation used in this paper.

3.4 Methodology

3.4.1 Rationale

In this subsection, we explain the rationale behind our work. In the MAB field, the agent’s goal is to maximize the total reward in the long run so it needs exploration besides exploitation [51]. It is undeniable that when the user of a system is a human, the exploration is very
Table 3.1: Notation Description

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}$</td>
<td>User Profile unit</td>
</tr>
<tr>
<td>$\mathcal{K}$</td>
<td>Knowledge-base</td>
</tr>
<tr>
<td>$\mathcal{I}_u$</td>
<td>User interests</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of items</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of users</td>
</tr>
<tr>
<td>$T$</td>
<td>Number of time steps</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of articles to recommend in a single time step</td>
</tr>
<tr>
<td>$C$</td>
<td>Number of categories in the knowledge-base</td>
</tr>
<tr>
<td>$r$</td>
<td>Numerical reward</td>
</tr>
<tr>
<td>$R$</td>
<td>Cumulative reward in $T$ time steps</td>
</tr>
<tr>
<td>$a$</td>
<td>An arm</td>
</tr>
<tr>
<td>$A$</td>
<td>The set of arms</td>
</tr>
<tr>
<td>$\eta$</td>
<td>The number of possible user interests</td>
</tr>
<tr>
<td>$\delta$</td>
<td>The epoch for change in the user interest</td>
</tr>
</tbody>
</table>

costly [54]. This is mainly because the human is complex in nature and can become bored so quickly when recommendations are irrelevant. Therefore, our first and foremost goal is to use a method that needs minimum exploration, or better say, that explores whenever necessary while achieving the maximum satisfaction in the long-term. Secondly, it is clear that if the users know and say what specifically is of their interest, then the recommendation task becomes very straightforward. However, the research indicates that the users are too lazy to provide necessary feedback on recommendations received [55]. Moreover, in many cases, the users are not clear about their interests [56]. Accordingly, we think a good recommendation method should need minimum or no information from users, and if it needs, it should implicitly learn them through interacting with the user. Finally, in the healthcare field, it is of major importance to recommend accurate and reliable information to patients [21, 22]. Otherwise, it can have costly consequences. So, typically the best way to recommend the most reliable information to users is to use KBs. Regarding these facts, we have proposed KERS — a hybrid method that utilizes an on-demand exploration scheme, described below.
3.4.2 Proposed KERS

Fig. 3.1 illustrates the architecture of KERS. KERS is composed of two phases: Exploration and Exploitation. In the Exploration phase, since the idea is to find user interests (i.e., topics or categories the user is interested to know about and indicated by \( \mathcal{I}_u \)), KERS recommends one random article from each category. When the user clicks on an article’s title, the category of that article is added to \( \mathcal{I}_u \). User Profile (\( \mathcal{P} \)) unit (see Fig. 3.1) is responsible to keep the track of users and their interests by saving their IDs and \( \mathcal{I}_u \). It is noteworthy to mention that if \( C \) is larger than \( k \), Exploration might take a couple of time steps to complete. For instance, if \( k = 5 \) and \( C = 8 \), Exploration takes two time steps, e.g., \( t_1 \) and \( t_2 \), to complete. At \( t_1 \), five articles from first five categories are recommended. At \( t_2 \), three articles from the remaining three categories plus two articles from random categories are recommended (i.e., these two articles could be from any of eight categories). Note that, at \( t_2 \), we could only recommend three articles from the remaining three categories; however, we aim to preserve the consistency and recommend exactly \( k \) articles at each time step. In the Exploitation phase, KERS uses \( \mathcal{P} \) and randomly recommends \( k \) articles from \( \mathcal{I}_u \). Note that if \( |\mathcal{I}_u| > 1 \) for a user, KERS tries to fairly recommend \( k \) articles from all these categories. More specifically, if a user is interested in \( \eta \) topics, KERS recommends \( k/\eta \) articles from \( C_1 \), \( k/\eta \) from \( C_2 \), and so on. If \( k \) is not divisible by \( \eta \), KERS uses a rounding up method; e.g., if \( k = 5 \), \( \eta = 2 \), and \( C = 5 \), it recommends three articles from \( C_1 \) and two articles from \( C_2 \). Overall, in the first time step(s), KERS is in the Exploration phase and recommends articles from all categories. Then, when Exploration completes, it switches to the Exploitation phase and remains there until \( \mathcal{I}_u \) changes. KERS detects this change through receiving no reward in a time step. Once occurred, KERS switches back to Exploration to find out new \( \mathcal{I}_u \). This process is repeated until time step \( T \) or when the user quits the system. The pseudo code of KERS is presented in Algorithm 1.
**Algorithm 1: KERS algorithm**

1. **initialization** ($t = 1$, $\mathcal{I}_u = \emptyset$, $phase = \text{Exploration}$, $exp\_ctr = 0$)

2. while $t <= T$ do

3.  

4.  

5.  

6.  

7.  

8.  

9.  

10.  

11.  

12.  

13.  

14.  

15.  

16.  

17.  

18.  

19.  

20.  

21.  

22.  

23.  

24.  

25.  

26.  

27.  

28. end
3.5 Experiments

In this section, the performance of KERS is validated through simulation study. First the experiments setup is described. Then, the results of experiments are presented.

3.5.1 Setup

Parameters and Baselines

In order to investigate the effect of different values of $M$ on the performance of the system, we have picked the following values: 1, 10, and 100. If $M > 1$, in every time step, one user is randomly selected as the active user by the system and he receives the recommendation. There are 10,000 time steps ($T$) and every experiment is the average over 20 runs. We pick $k = 5$ to avoid choice overload [57] and use $R$ as the performance metric. For performance comparison, $\epsilon$-greedy and UCB1 are implemented as the baseline algorithms. Although, for the sake of clarity, we only report the results for $\epsilon$-greedy when $\epsilon = 0.1$, KERS outperforms $\epsilon$-greedy irrespective of the value of $\epsilon$. 
Datasets

Two datasets are used in our experiments: prostate cancer and BBC [58]. We have developed a dataset with 500 articles ($N = 500$) about prostate cancer, which are crawled from prostate cancer websites\(^3\). The reliability of the content of these websites have been validated by our experts and the level of information is understandable to a layperson. The articles are categorized into five groups: *pre-diagnosis, diagnosis, treatment, side effects*, and *recurrence issues*. Ref. [59] inspired us for this classification and, with the advice from our experts, we extended it to five categories, which better reflects the current informational needs of prostate cancer patients observed in clinical communications. To see the performance on a larger dataset, we have also used BBC dataset in our experiments. The articles in this dataset are categorized into five topics, i.e., *business, entertainment, politics, sport*, and *technology*, and there are 2225 articles in the dataset ($N = 2225$).

User Simulator

According to Ref. [5], the user’s interest is dynamic and changes over time. This change is sporadic and depends on many factors, including personality, mood, and time. Accordingly, it is almost infeasible to exactly model a user. Instead, in this paper, we present a simple user simulator and use it to evaluate our method. Basically, the user interest has two main elements: the number of topics a user might be interested to know about ($\eta$) and the time or duration the user is interested about them ($\delta$). For example, a patient may be interested in the topics of diagnosis and treatment, as well as he might become bored after some time (e.g., 10 minutes) and leave the platform. Upon returning to use the platform again, he might become interested in side effects and recurrence issues topics. For simplicity, we assume that these two factors are fixed in our users. In other words, if $\eta = 2$ and $\delta = 20$ time steps for a user, he stays interested in two topics until $t = T$, but he might change these two topics

---

\(^3\)Prostate Cancer Canada, Canadian Cancer Society, Canadian Urological Association, American Urological Association, Wikipedia
every 20 time steps. We let $\eta \in [1, 2, 3]$ and $\delta \in [10, 20, \ldots, 100]$. That means, one user may change his interest every 10 time steps and another one every 20 time steps. With this setting, a user can stay focused on merely one topic all the time. The system assigns $\eta$ and $\delta$ to the users randomly. Note that users can have similar behaviors; it is quite possible that two users have similar $\eta$ and $\delta$.

### 3.5.2 Results

**Prostate cancer dataset**

Fig. 3.2 shows $R$ for the three algorithms when the values of $M$ and $\eta$ vary from 1 to 100 and 1 to 3, respectively. In general, KERS outperforms the two methods in all settings. This superiority is more tangible when $M$ and $\eta$ are smaller. More specifically, when $M = 10$ and $\eta = 1$, KERS outperforms the baseline methods by 100% (Fig. 3.2(d)). The reason of this superiority is that KERS finds $I_u$ as quickly as possible and exploits this knowledge in subsequent time steps. KERS returns to the Exploration phase only if it sees a change in $I_u$. While the baseline methods perform greedily in most of the time, their exploration is not wise enough. UCB1 performs relatively better than $\epsilon$-greedy when $\eta$ is smaller (see Figs. 3.2(a) and 3.2(b)). As $M$ increases, predicting $I_u$ becomes more difficult and the performance of the three methods diminishes accordingly. Whereas KERS achieves almost the perfect $R$ (i.e., 50,000 after 10,000 time steps) when $M = 1$ and $\eta \in [1, 2, 3]$, this number becomes about 15,000 when $M = 100$ (Figs. 3.2(g), 3.2(h), and 3.2(i)). This is mainly because when there are more users in the system and their interest changes after a while, more exploration is required to find the current interest of users and exploitation becomes ineffective. Another observation is that when $\eta$ increases, $R$ is better in all methods. This makes sense as when the users are interested in multiple topics simultaneously, it is easier to predict their interests compared to when they are interested in only one topic.
Figure 3.2: The cumulative reward for the three algorithms on prostate cancer dataset

**BBC dataset**

Fig. 3.3 shows $R$ for the three algorithms applied to the BBC dataset. Again, KERS outperforms the baseline methods in all settings and the same observations about change in $M$ and $\eta$ and their effect on the performance of the algorithms are seen here. The only difference is that UCB1 performs the worst among the three algorithms. The reason is that this method should explore and play all arms first and then starts to balance exploration vs. exploitation. Because there are more arms to explore in this dataset and the users interests also change over time, UCB1 performs the worst.
Finally, Fig. 3.4 illustrates the number of times each arm in the BBC dataset is played by each algorithm. To be clear, we have shown the results when $M = 1$, $\delta = T$ (i.e., the user interest does not change over time), and $\eta = 1$ (the user is interested in the topic of technology). As depicted, both KERS and UCB1 fairly recommend all articles in this topic. On the other hand, $\epsilon$-greedy performs quite poorly in this term as it greedily recommends a few articles all the time. For example, it recommends one article in this topic more than 9000 times. This clearly shows while $\epsilon$-greedy performs relatively better in this dataset in terms of $R$ compared to UCB1, it is almost impractical for recommending articles to a human.
3.5.3 Discussion

KERS borrows the advantages of the two methods, i.e., KBRSs and MABs, as well as it covers their problems. In other words, while it is simple and practical, it alleviates the exploration vs. exploitation problem. At the same time, it is content (articles’ body text) and dataset size independent and can provide reliable recommendations. However, as all knowledge-based systems, one might argue that the performance of KERS depends on the accuracy of the KB used and its performance degrades if the KB is not carefully developed. This dependence on the KB may also have a detrimental effect on the exploration time in case the number of categories in the KB is very large, although it is rare.

Patients diagnosed with prostate cancer may be asked by their clinician to share in decisions regarding treatment, given the inherent trade-offs across the treatment options associated with adverse quality of life outcomes. To make informed decisions, these patients may have informational needs that cannot be fulfilled with conventional resources. The personalization supplied by KERS could be used to provide prostate cancer patients with more useful information compared to existing educational resources (e.g., patient pamphlet). This, in turn, could help better inform patients about prostate cancer, the treatments that
are available to them, and better inform their expectations over the duration of their care pathway.

3.6 Conclusion and Future Work

In this paper, we have proposed KERS — a knowledge-based exploration on-demand RS algorithm for cancer patients information provisioning. Since exploration is expensive when the user of the system is a human, the main objective of KERS is to achieve the maximum long-term satisfaction through minimum exploration. The results of experiments have confirmed the effectiveness of KERS compared to baseline algorithms.

This paper is a pilot study to initially test the performance of KERS offline and in a small scale. We have two plans in our future work agenda. First, we plan to take one step further and make KERS context-aware. Second, we will test the performance of KERS online, i.e., using prostate cancer patients, on a large dataset.
Reinforcement Learning based Recommender Systems: A Survey

Abstract

Recommender systems (RSs) have become an inseparable part of our everyday lives. They help us find our favorite items to purchase, our friends on social networks, and our favorite movies to watch. Traditionally, the recommendation problem was considered to be a classification or prediction problem, but it is now widely agreed that formulating it as a sequential decision problem can better reflect the user-system interaction. Therefore, it can be formulated as a Markov decision process (MDP) and be solved by reinforcement learning (RL) algorithms. Unlike traditional recommendation methods, including collaborative filtering and content-based filtering, RL is able to handle the sequential, dynamic user-system interaction and to take into account the long-term user engagement. Although the idea of using RL for recommendation is not new and has been around for about two decades, it was not very practical, mainly because of scalability problems of traditional RL algorithms. However, a new trend has emerged in the field since the introduction of deep reinforcement learning (DRL), which made it possible to apply RL to the recommendation problem with large state and action spaces. In this paper, a survey on reinforcement learning based rec-
ommender systems (RLRSs) is presented. Our aim is to present an outlook on the field and to provide the reader with a fairly complete knowledge of key concepts of the field. We first recognize and illustrate that RLRSs can be generally classified into RL- and DRL-based methods. Then, we propose an RLRS framework with four components, i.e., state representation, policy optimization, reward formulation, and environment building, and survey RLRS algorithms accordingly. We highlight emerging topics and depict important trends using various graphs and tables. Finally, we discuss important aspects and challenges that can be addressed in the future.

4.1 Introduction

We are living in the Zettabyte Era [60]. The massive volume of information available on the web leads to the problem of information overload, which makes it difficult for a decision maker to make right decisions. The realization of this in our everyday lives is when we face a long list of items in an online shopping store; the more items in the list, the tougher it becomes to select among them. Recommender systems (RSs) are software tools and algorithms that have been developed with the idea of helping users find their items of interest, through predicting their preferences or ratings on items [61, 16]. In fact, the idea is to know the users to some extent, i.e., making a user profile based on their feedback on items, and to recommending those items that match their profile. Today, RSs are an essential part of most giant companies, like Google, Facebook, Amazon, and Netflix, and employed in a wide range of applications, including entertainment [9, 10, 11], e-commerce [12], news [13], e-learning [14], and healthcare [15].

Numerous techniques have been proposed to tackle the recommendation problem; traditional techniques include collaborative filtering, content-based filtering, and hybrid methods. Despite some success in providing relevant recommendations, specifically after the introduction of matrix factorization [62], these methods have severe problems, such as cold start, lack
of novelty and diversity, scalability, low quality recommendation, and great computational
expense [61, 16, 63]. Recently, deep learning [64] has also gained popularity in the RS field
due to its ability in finding complex and non-linear relationships between users and items
and its cutting edge performance in recommendation [17]. Nonetheless, deep learning models
are usually non-interpretable, data hungry (this is specifically problematic as the amount of
data, i.e. rating/user feedback, in the RS field is scarce), and computationally expensive [17].

RL is a semi-supervised machine learning field in which the agent optimizes its behavior
through interaction with the environment. The milestone in the RL field is the combination
of deep learning with traditional RL methods, which is known as deep reinforcement learning
(DRL) [65, 66]. This made it possible to apply RL in problems with enormous state and
action spaces, including self-driving cars [67, 68], robotics [69], industry automation [70],
finance [71], healthcare [72, 73], and RSs [7]. The unique ability of an RL agent in learning
from a reward from the environment without any training data makes RL specifically a
perfect match for the recommendation problem. Today, more and more companies are
utilizing the power of RL to recommend better items to their customers. For example, in
a study by researchers at Google [6], it is shown that RL can be employed to recommend
better video content to YouTube’s users. In fact, the use of RL in the RS community is not
limited to the industry, but it is becoming a trend in academia as well. Fig. 4.1(a) illustrates
this trend.

This trend and this topic motivated us to prepare this survey paper, which aims at
providing a comprehensive overview of the state-of-the-art in reinforcement learning based
recommender systems (RLRSs). Our main purpose is to depict a high-level picture from
the progress in the field since the beginning and show how this trend has been significantly
changed with the advent of DRL. At the same time, we provide detailed information about
each method in the form of tables so as the reader can easily observe the similarities and
differences between methods.

**Paper Collection Methodology.** To collect relevant papers, we have used a multi-level
Figure 4.1: Publications information of 97 surveyed RLRS papers. (a) Distribution of RLRSs publications per year (until September 2021) separated based on RL and DRL methods. (b) Venue distribution of published RLRSs. To be clear, we filtered venues with only one publication and termed them as ‘Others’ in the graph. This includes a long list of venues, including AAMAS, ICML, ICDM, and JMLR. (c) The proportion of different types of publications, i.e., conference proceedings, journal articles, and arXiv preprints, of surveyed RLRSs.
search process. The focus of this survey paper is specifically on RSs that use an RL algorithm. Accordingly, in order to find relevant papers, we used Google Scholar as the main search engine and searched “reinforcement learning recommender system” keyword. This search resulted in around 33,000 papers. Out of first 1000 articles found, we collected 500 papers as the result of our first screening level. Then, to increase the reliability of our paper collection, we also explored related libraries like ACM digital library, IEEE Xplore, SpringerLink, and ScienceDirect with the same keyword and until a point where no more relevant paper was among the results. We found that all related articles identified in these libraries were available in our initial search using Google Scholar. With carefully studying collected articles and excluding irrelevant papers, duplicates, theses, survey and review papers, we selected 97 articles to include in our survey paper. Although we are certain that we did not find all RLRSs through the search process explained, we are confident that we could find the vast majority of relevant publications.

It is noteworthy to mention that we did not include RSs based on multi-armed bandits. Bandits are a simplified version of RL. In particular, in bandits, similar to an RL problem, the agent should learn to maximize a numerical reward through interaction with the environment and solving the exploration vs exploitation dilemma [74, 25]. However, different from full RL, bandits are stateless; that means, learning happens in only one state. While bandits have been popular for the recommendation problem [52, 75, 76, 77], in light of recent successful applications of DRL and unprecedented interests in full RL by the RS community, we opt to only focus on RSs that use a full RL algorithm. The curious reader is referred to a recent survey on the application of bandits to RSs [78].

**Related Work.** Plenty of research has been done in the field of RSs and a plethora of survey papers have been published, including RSs [63], collaborative filtering [79, 80], hybrid methods [81], multi-media RSs [82, 83], explainable recommendation [84], and article RSs [85], to name a few. There are also some published survey papers on topics closely related to RLRSs [86, 17, 87, 88]. Perhaps two closest survey papers to ours are [86, 17]. Ref. [86]
surveys DRL-based information seeking techniques, such as search, recommendation, and online advertising. Authors discuss several RSs, which use multi-armed bandits and DRL for policy optimization. However, the work misses many important RLRSs and fails to provide an in-depth analysis of algorithms reviewed. Zhang et al. [17] provide a comprehensive survey on deep learning based RSs. They consider DRL as a deep learning architectural paradigm and survey a few DRL-based RSs [89, 90, 91, 92, 5, 2]. Nonetheless, this classification is not correct and DRL is not a deep learning architecture, but it is an extension to traditional RL algorithms. Other related surveys target sequence-aware [87] and session-based [88] recommendation techniques. Ref. [87] surveys sequence-aware RSs. Authors consider RL as a method for sequence learning and review some RL-based RSs [93, 19]. In another related survey [88], RL is considered as a method for session-based RSs [94, 95, 92]. None of previous published survey papers provide a comprehensive overview and in-depth analysis of published RLRSs. To the best of our knowledge, this is the first survey paper that specifically targets RLRSs.

Our contribution. The goal is to provide the reader with a vista toward the field so that they can quickly understand the topic and major trends and algorithms presented so far. This helps researchers see the big picture, compare algorithms’ strengths and weaknesses, and shed some light on ways to advance them in the future. Our main contributions can be summarized as:

- Presenting a framework for RLRSs. We first generally divide RLRSs into RL- and DRL-based methods. Then, we propose a framework with four main components, i.e., state representation, policy optimization, reward formulation, and environment building. This framework can model every RLRS and unify the development process of RLRSs.

- Providing a thorough background on RL. We provide the reader with a fairly complete knowledge on RL/DRL and their various algorithms used by RLRSs.
• Highlighting important trends and emerging topics. Instead of simply summarizing algorithms, our aim is to extract and to illustrate major trends and attempts in each main component of the proposed framework in particular and emerging topics in RLRSs in general.

• Suggesting some open research directions for the future. In order to consolidate our survey paper, we finally present some observations about ongoing research in the RLRS field and propose some open research directions to advance the field.

The remaining of this paper is organized as follows. In section 4.2, to help the reader better understand the topic, we discuss some preliminary concepts and provide a solid background on RL. Section 4.3 presents RLRSs algorithms in a classified manner. Emerging topics are highlighted in section 4.4. In section 4.5, some open research directions are suggested for the future work, and finally, the paper is concluded in section 4.6.

4.2 Preliminaries

In this section, we provide a background on the important concepts discussed throughout this paper. This background provides the reader with useful and concise information about RSs, RL and DRL, why it is necessary to use RL in RSs, problem formulation, and the proposed framework for RLRSs.

4.2.1 Recommender Systems

In everyday life, it is not very uncommon to face situations in which we have to make decisions while we have no a priori information about options. In such a case, relying on recommendations from others, who are experienced in that aspect, seems quite necessary [96]. This was the rationale behind the first RS, Tapestry [97], and authors termed it as collaborative filtering. Later, this term was broadened to recommender systems to reflect two facts [96]:

37
1) the method may not be based on an implicit collaboration between users, 2) the method may suggest interesting items, not filter them. By definition, RSs are software tools and algorithms that suggest items that might be of interest to the users [16]. Another important approach toward the recommendation problem is content-based filtering, in which the idea is to use items descriptions and devising a method to match them with the user profile, a structured representation of user interests [98, 99]. Collaborative filtering usually suffers from data sparsity, scalability, and gray sheep [79]. Content-based filtering also has some shortcomings, including limited content analysis, serendipity, and new user [99]. Hybrid methods, a combination of the two, can only alleviate part of these problems [16, 79].

4.2.2 From Reinforcement Learning to Deep Reinforcement Learning

Reinforcement learning (RL) is a machine learning field that studies problems and their solutions in which agents, through interaction with their environment, learn to maximize a numerical reward. According to Sutton and Barto [25], three characteristics distinguish an RL problem: (1) the problem is closed-loop, (2) the learner does not have a tutor to teach it what to do, but it should figure out what to do through trial-and-error, and (3) actions influence not only the short-term results, but also the long-term ones. The most common interface to model an RL problem is the agent-environment interface, depicted in Fig. 4.2. The learner or decision maker is called agent and the environment is everything outside the agent. Accordingly, at time step $t$, the agent sees some representations/information about the environment, called state, and based on the current state it takes an action. On taking this action, it receives a numerical reward from the environment and finds itself in a new state.

More formally, the RL problem is typically formulated as a Markov decision process (MDP) in the form of a tuple $(S, A, R, P, \gamma)$, where $S$ is the set of all possible states, $A$ is the set of available actions in all states, $R$ is the reward function, $P$ is the transition
probability, and $\gamma$ is the discount factor.

The main elements of an RL system are [25]:

- **Policy**: policy is usually indicated by $\pi$ and gives the probability of taking action $a$ when the agent is in state $s$. Regarding the policy, RL algorithms can be generally divided into on-policy and off-policy methods. In the former, RL methods aim at evaluating or improving the policy they are using to make decisions. In the latter, they improve or evaluate a policy that is different from the one used to generate the data.

- **Reward signal**: upon selecting actions, the environment provides a numerical reward to inform the agent how good or bad are the actions selected.

- **Value function**: the reward signal is merely able to tell what is good immediately, but the value function defines what is good in the long run.

- **Model**: model provides the opportunity to make inferences about the behavior of the environment. For instance, the model can predict next state and next reward in a given state and action [25].

**Algorithms.** Many algorithms have been proposed to solve an RL problem; they can be generally divided into tabular and approximate methods [25]. In tabular methods, since the size of action and state spaces is small, value functions can be represented as tables and
optimal value function and policy can be found. On the other hand, in approximate methods, since the size of state space is enormous, the goal is to find a good approximate solution with the constraint of limited computational resources. As mentioned earlier, with the foundation of DRL, a substantial change has emerged in the RL field in general. Accordingly, although DRL belongs to the approximate group, we generally divide RL algorithms used by RLRSs into RL-based and DRL-based algorithms as we believe this classification better reflects the recent trend in the RLRS field. It is noteworthy to mention that the distinguishing factor between DRL and traditional RL algorithms is that DRL algorithms use deep learning for function approximation (a more detailed explanation on this is presented shortly in DRL-based Algorithms section). In the following, we briefly review those RL algorithms employed by RLRSs. Fig. 4.3 illustrates these algorithms.

1) RL-based Algorithms. As stated before, RL algorithms could be divided into tabular and approximate methods. Popular tabular methods include dynamic programming, Monte Carlo, and temporal difference. Dynamic programming methods assume a perfect model of the environment and use a value function to search for good policies. Two important algorithms from this class are policy iteration and value iteration. Policy iteration algorithm composes of three steps: initialization, policy evaluation, and policy improvement. First the policy is randomly initialized, i.e., a random action \( a \in A(s) \) is selected for all \( s \in S \). Then,
the value of the states are computed and evaluated using

\[ V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) \left[ r + \gamma V(s') \right], \]  

(4.1)

where \( p \) is the transition probability and \( s' \) is the next state. Finally, the policy is updated as follows, \( \forall s \in S \):

\[ \pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r|s, a) \left[ r + \gamma V(s') \right]. \]  

(4.2)

As pointed out in [25], a problem with policy iteration algorithm is that it needs policy evaluation in every iteration, which can be computationally prohibitive. **Value iteration** algorithm is a special case of policy iteration algorithm in which policy evaluation is stopped after one sweep. More precisely, \( V(s) \) is randomly initialized \( \forall s \in S \). Then, it is updated in each step according to

\[ V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[ r + \gamma V(s') \right]. \]  

(4.3)

Unlike dynamic programming, **Monte Carlo** methods do not need a complete knowledge assumption about the environment. They only need a sample sequence of states, actions, and rewards from the environment, which could be real or simulated. Monte Carlo Tree Search (MCTS) is an important algorithm from this family. The basic **MCTS** is an iterative search tree building algorithm that runs until reaching a predefined computational budget [100]. It composes of four steps: selection, expansion, simulation, and backpropagation [101]. Starting at the root node, a recursive child selection policy is executed to select the leaf node (expandable node). The tree is then expanded by adding one or more child nodes. Next, a simulation of a complete episode is performed from newly added leaf nodes to produce an outcome. Finally, the result of simulation is back propagated in the tree (through selected nodes). Later in 2016, MCTS was merged with deep learning [102].

**Temporal difference** methods are a combination of dynamic programming and Monte
Carlo methods. While they do not need a model from the environment, they can bootstrap, which is the ability to update estimates based on other estimates [25]. From this family, Q-learning [103] and Sarsa [104] are very popular. **Q-learning** is a model-free, off-policy algorithm to learn the value of an action in a given state. The main component of Q-learning is the following *Bellman* equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)],$$

(4.4)

where $\alpha$ is the learning rate. **Sarsa** is an online version of Q-learning with the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)].$$

(4.5)

**Sarsa($\lambda$)** employs *eligibility traces*, which unify temporal difference and Monte Carlo methods [25]. In Sarsa($\lambda$), the weight vector is updated on each step using

$$w_{t+1} = w_t + \alpha \delta_t e_t,$$

(4.6)

where $\delta_t$ is temporal difference error and defined by

$$\delta_t = r_{t+1} + \gamma q(s_{t+1}, a_{t+1}, w_t) - q(s_t, a_t, w_t),$$

(4.7)

where $q$ is the approximate action value. Also, the eligibility trace $e_t$ is defined as

$$e_t = \gamma \lambda e_{t-1} + \nabla q(s_t, a_t, w_t), \quad 0 \leq t \leq T, e_{-1} = 0$$

(4.8)

where $\lambda \in [0, 1]$. More details on Sarsa($\lambda$) can be found in [25]. **R-learning** [105] is an extension of Q-learning for continuing tasks where the interaction between agent and environment continues forever without termination. The main idea in R-learning is the concept
of average reward; that means, discounted reward setting is not necessary, even problematic, in continuous function approximation and there is no difference between immediate and delayed rewards [25]. That said, Q-learning equation, Eq. (4.4), could be rewritten as [93]

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r - \rho + \max_{a'} Q(s', a') - Q(s, a)], \quad (4.9)$$

where $\rho$ is the average reward and can be achieved by

$$\rho \leftarrow \rho + \beta [r - \rho + \max_{a'} Q(s', a') - \max_a Q(s, a)], \quad (4.10)$$

where $\beta$ is a learning rate.

In approximate methods, a practical approach is to generalize from previous experiences (already seen states) to unseen states. Function approximation is the type of generalization required in RL and many techniques could be used to approximate the function, including artificial neural networks. Fitted Q algorithm is an approximate method that is inspired by the idea of fitted value iteration proposed by Gordon [106]. An interesting fact about fitted Q framework is that it allows to use any regression algorithm for function approximation. For instance, Ernst et al. [107] use randomized trees for function approximation and call their method fitted Q iteration (FQI). Among approximate solutions, policy gradient methods have been very popular, which learn a parameterized policy and can select actions without the need of a value function. REINFORCE [108] is a Monte Carlo method that uses episode samples in order to update the policy parameter $\theta$. It first randomly initializes $\theta$. Then, it iteratively generates a trajectory following policy $\pi_\theta : S_1, A_1, R_1, ..., S_T$. For each step $t = 1, 2, ..., T$, it estimates return $G_t$ and updates $\theta$ using

$$\theta \leftarrow \theta + \alpha \gamma t G_t \nabla \log \pi(A_t | S_t, \theta). \quad (4.11)$$

A major problem with REINFORCE is that it has high variance in gradient estimation. To
overcome this problem, a baseline (state-value function) is added to the update rule:

$$\theta \leftarrow \theta + \alpha \gamma^t \left(G_t - b(S_t)\right) \nabla_\theta \log \pi(A_t|S_t, \theta).$$ \hspace{1cm} (4.12)

This method is called REINFORCE-with-baseline (REINFORCE-wb) [25]. While REINFORCE-wb reduces the variance, it is still a Monte Carlo method and has a slow convergence. A better solution is to add bootstrapping to REINFORCE; an idea employed by actor-critic method [109]. In particular, instead of a baseline, a critic is used to criticize the policy generated by the actor. That said, the update rule for actor-critic is revised as follow

$$\theta \leftarrow \theta + \alpha \gamma^t \left(G_t - v(S_t, w)\right) \nabla_\theta \log \pi(A_t|S_t, \theta),$$ \hspace{1cm} (4.13)

where $v(s, w)$ is a state-value function parameterized with $w$.

2) DRL-based Algorithms. DRL is an interesting combination of deep learning with RL. In fact, researchers at DeepMind found that this combination can achieve human-level performance in Atari games [110, 111]. Deep Q-network (DQN) is the first DRL algorithm introduced [110], which is a creative combination of convolutional neural networks (CNN) [64] with Q-learning. More precisely, in DQN, a Q network is responsible for action-value approximation, which could be trained to minimize the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right],$$ \hspace{1cm} (4.14)

where $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$ is the target for iteration $i$ and $\rho$ is a probability distribution over transitions $s, a, r, s'$ collected from the environment. Differentiating $L(\theta)$ in Eq. (4.14) with respect to $\theta$ yields the following gradient

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right].$$ \hspace{1cm} (4.15)
It is computationally beneficial to optimize the gradient in Eq. (4.15) using stochastic gradient descent [111]. According to [25], DQN modifies the original Q-learning algorithm in three ways: 1) It uses experience replay, first proposed in [112] and a method that keeps agents’ experiences over various time steps in a replay memory and uses them to update weights in the training phase. 2) In order to reduce the complexity in updating weights, current updated weights are kept fixed and fed into a second (duplicate) network whose outputs are used as Q-learning targets. 3) The error term \((r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\) in Eq. (4.15) is clipped such that it remains in the interval \([-1, 1]\). All these modifications help improve the stability of DQN.

However, DQN has some problems; first, following Q-learning algorithm, DQN overestimates action values under certain circumstances, which makes learning inefficient and can lead to sub-optimal policies [113]. Double DQN (DDQN) was proposed to alleviate this problem [114]. The difference between DQN and DDQN is that the greedy policy is evaluated using online network, but the target network is used to estimate its value. Thus, \(y_i\) is changed as follows

\[
y_i = r + \gamma Q\left(s', \arg \max_{a'} Q(s', a'; \theta_i); \theta_{i-1}\right).
\] (4.16)

An interesting extension on top of DDQN is dueling network [115] whose idea is to have a single Q network with the same convolutional layers as DQN, but to have two streams of fully connected (FC) layers, which provide estimates for value and advantage functions. This helps better generalize learning between actions. Second, DQN uniformly selects experiences to replay regardless of their significance, which makes the learning process slow and inefficient. Accordingly, prioritized experience replay was proposed to solve the problem [116]. The idea is to replay important experiences more often, so as the network training is improved. The importance of each transition is measured proportional to temporal difference error, and two variants, stochastic prioritization and importance sampling, are proposed to improve it. Finally, DQN is not applicable in continuous spaces, so deep deterministic policy gradient (DDPG) [117] was proposed, which is a combination of DQN and deterministic policy
gradient (DPG) [118] in an actor-critic approach. Actor (DPG module) deterministically maps states to a specific action. Critic (DQN module) defines the value of the action taken by actor. In every iteration, critic is updated by

\[ L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \]  

(4.17)

and actor is updated by

\[ \nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q) | (s = s_i, a = \mu(s_i)) \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_i, \]  

(4.18)

where \( \theta^\mu \) and \( \theta^Q \) are the parameters of actor and critic networks, respectively.

Finally, proximal policy optimization (PPO) [36] is another actor-critic algorithm used by RLRSs. In fact, PPO is an improved version of trust region policy optimization (TRPO) [119] algorithm, which maximizes a surrogate objective

\[ \mathbb{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right], \]  

(4.19)

where \( A_t \) is an estimator of advantage function at \( t \). The core idea in PPO is the introduction of clipped surrogate objective, as

\[ \mathbb{E}_t \left[ \min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t, \text{clip} \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right], \]  

(4.20)

where \( \epsilon \) is a hyperparameter.

**RL Challenges.** There are some possible challenges when applying RL to any problem. A challenge well-known as **Deadly Triad** states that there is a hazard of instability and divergence when combining three elements in RL: function approximation, bootstrapping, and off-policy training [25]. Another challenge in RL is **sample inefficiency**, specifically in model-free RL algorithms. Current model-free RL algorithms need a considerable amount of
agent-environment interaction in order to learn useful states. Moreover, since DRL is based on deep learning, it consequently inherits the famous feature of neural networks, i.e., being black-box. It is not obvious how weights and activations are changed, which makes them uninterpretable. The classical problem of exploration vs exploitation is still a challenge in RL and effective exploration is an open research problem. Finally, the problem of reward formulation in RL is a challenge and designing a good reward function is not very clear or straightforward.

4.2.3 Why Reinforcement Learning for Recommendation?

The nature of user interaction with an RS is sequential [18] and the problem of recommending the best items to a user is not only a prediction problem, but a sequential decision problem [19]. This suggests that the recommendation problem could be modelled as an MDP and be solved by RL algorithms. Three unique features of RL make it a perfect match for the recommendation problem. First, RL is able to handle the dynamics of sequential user-system interaction by adjusting actions according to continuous feedback received from the environment. Second, RL is able to take into account the long-term user engagement with the system. Finally, although having user ratings is beneficial, RL, by nature, does not need user ratings and optimizes its policy by sequentially interacting with the environment. All these reasons suggest that it would be beneficial to use RL to provide better recommendations, as proven by online studies [6, 120].

4.2.4 Problem Formulation

In a recommendation problem, the RS algorithm, through interaction with the user and receiving their implicit/explicit feedback, tries to recommend the best items to the user, in order to achieve the goal it is designed for, which could be increasing profit, user satisfaction, or user fidelity [16]. This is analogous to a typical RL setting, where an agent aims at maximizing a numerical reward through interaction with an environment [25]. Therefore,
the RL agent can play the role of the RS algorithm and every thing outside this agent, including the users of the system and items, can be considered as the environment for this agent.

More formally, considering the user and items as the environment and the RS algorithm as the RL agent, MDP formulation can be as follows:

- **State** $S$: a state $s_t \in S$ is defined as the user preferences and their past history with the system.
- **Action** $A$: an action $a_t \in A$ is to recommend an item to the user at time step $t$.
- **Reward** $R$: the RL agent receives reward $r(s_t, a_t) \in R$ based on the user feedback on the recommendation provided.
- **Transition probability** $P$: transition probability $p(s'|s, a) \in P$ is the probability of transition from $s = s_t$ to $s' = s_{t+1}$ if action $a$ is taken by the agent.
- **Discount factor** $\gamma$: discount factor $\gamma \in [0, 1]$ is the discount factor for future rewards. With $\gamma = 0$, the agent becomes myopic, i.e., it only focuses on immediate reward. On the contrary, if $\gamma = 1$, the agent becomes farsighted and focuses more on future rewards [25].

Given $(S, A, R, P, \gamma)$, the goal of the RL agent is to find a policy $\pi$ that maximizes the expected, discounted cumulative reward. In other words,

$$
\max_{\pi} \mathbb{E}\left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right],
$$

where $T$ is the maximum time step in a finite MDP.

### 4.2.5 Proposed RLRS Framework

With carefully studying all RLRSs collected, we found that there are four components common in all of them and believe that a good RLRS should carefully design and address these components. Accordingly, to unify the process of RLRS development, we propose a
Figure 4.4: The proposed RLRS framework

framework for RLRSs with four key components: (1) State Representation, (2) Policy Optimization, (3) Reward Formulation, and (4) Environment Building. Fig. 4.4 depicts this framework. In the following, we explain each component.

**State Representation.** In the agent-environment RL interface, the state can be *any* information available to the agent. State representation could be as high-level as symbolic descriptions of objects in a room or as low-level as sensor readings [25]. What is important is that defined states should have the *Markov property*. That means, the state signal is not supposed to convey all the information about the environment to the agent, but it should summarize past information such that all relevant information is not missed. A state signal with this property is called Markov. In general, selecting state representation is currently more art than science [25].

In RLRSs, state representation should summarize information about users, items, and the context. We divide state representation in RLRSs into three groups:

**SR1) Treating items as states.** When the item space is small, e.g., it includes several web pages in a website, it is possible to treat each item as a state. However, this approach is certainly not scalable when the item space grows large. To tackle the scalability problem in larger items spaces, researchers found that states could indicate a set of items previously rated/consumed by the user. Fig. 4.5(a) depicts this representation.

**SR2) Features from users, items, and context.** A popular way for state represen-
Figure 4.5: (a) SR1, (b) SR2, (c) SR3. The dashed Encoding module in SR3 means that some models merely use input embeddings as states.

The first step is to extract some features from users, items, and context, as shown in Fig. 4.5(b). User features include demographic information, such as age, race, and gender. Item features may include price, category, and popularity. Context features may include time, platform, and location.

**SR3) Encoded Embeddings.** For effective training, deep models in DRL-based RSs need states to be dense, low-dimensional vectors. Fig. 4.5(c) illustrates a general, popular framework for state representation in DRL-based methods. Typically, first user, items, and context features are translated into dense, low-dimensional, continuous vectors called *embeddings*. Then, for better training, this embedding could be encoded using a recurrent neural network (RNN) model, which can help the model learn user’s sequential preferences [2]. Gated recurrent units (GRU) is typically more popular than long short-term memory (LSTM) for the RNN module as it has fewer parameters and can achieve the same or better performance [121]. To focus on important parts of the input, some researchers also use an attention layer in the encoding module and add some weights to the encoded vectors. Finally, the encoded vectors are concatenated to yield the final state.

**Policy Optimization.** When states are formulated, it is the policy that determines which action to take (i.e., which items to recommend) in each state. For policy optimization, various RL algorithms have been utilized by RLRSs. Before the advent of DRL, RL
methods used by RLRSs could be generally classified into tabular and approximate methods. Tabular methods include policy iteration, Q-learning, Sarsa, Sarsa(\(\lambda\)), and R-learning. Approximate methods include fitted Q and gradient value iteration. On the other hand, DRL methods could be generally divided into three groups: value-based (DQN), policy gradient (REINFORCE and REINFORCE-wb), and actor-critic (DDPG and PPO) methods. A classification of these algorithms is shown in Fig. 4.3.

**Reward Formulation.** As mentioned earlier, the reward signal from environment reflects how good or bad the agent is performing through selecting actions. Therefore, designing informative reward signal is critical for success/learning of the agent. In fact, in RL, the reward signal is the only way to tell the agent what to do, not how to do it [25]. In general, defining a proper reward function is a hard problem and it is more of a trial-and-error or engineering process. There is no definite rule to design a good reward function in a specific problem. In RLRSs, we have observed two general trends in designing the reward function: (R1) the reward function is a simple, sparse numerical reward, or (R2) the reward is a function of one or several observations from the environment.

**Environment Building.** In general, evaluating RSs is difficult [56, 16]. As a result, building a suitable environment to properly train and evaluate the agent in RLRSs is challenging. To better distinguish between different environment building methods, we generally divide them into three groups: offline, simulation, and online. In offline method, the environment is a static dataset containing the ratings of some users on some items. A common practice in offline methods is to train the agent on the training data (usually 70-80\% of the data) and then test it on the remaining data. In simulation studies, usually a user model is built and the algorithm is evaluated while interacting with this user model. This user model could be as simple as a user with some pre-defined behavior, or it could be more complex and be learnt using available data. In online method, the algorithm is evaluated while interacting with real users and in real-time. This is the best, but most costly method for RLRSs evaluation.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAR</td>
<td>Average achievable rate</td>
<td>MIMIC</td>
<td>Multiparameter intelligent monitoring in intensive care</td>
</tr>
<tr>
<td>ACN</td>
<td>Average click number per capita</td>
<td>MJC</td>
<td>Mean Jaccard coefficient</td>
</tr>
<tr>
<td>ACS</td>
<td>Average click per session</td>
<td>ML</td>
<td>MovieLens</td>
</tr>
<tr>
<td>ADS</td>
<td>Average depth per session</td>
<td>MOOCs</td>
<td>Massive open online courses</td>
</tr>
<tr>
<td>AQ</td>
<td>Average quality</td>
<td>MR</td>
<td>Miss ratio</td>
</tr>
<tr>
<td>AR</td>
<td>Average reward</td>
<td>MRR</td>
<td>Mean reciprocal rank</td>
</tr>
<tr>
<td>ART</td>
<td>Average return time</td>
<td>MT</td>
<td>Movie tweetings</td>
</tr>
<tr>
<td>AT</td>
<td>Average turn</td>
<td>NDCG</td>
<td>Normalized discounted cumulative gain</td>
</tr>
<tr>
<td>AWT</td>
<td>Average watched tag per capita</td>
<td>NI</td>
<td>Number of user interactions before success</td>
</tr>
<tr>
<td>BC</td>
<td>BookCrossing</td>
<td>NV</td>
<td>Number of violated attributes</td>
</tr>
<tr>
<td>BDV</td>
<td>Book details viewed</td>
<td>OU</td>
<td>Ornstein-Uhlenbeck</td>
</tr>
<tr>
<td>BP</td>
<td>Number of books purchased</td>
<td>P</td>
<td>Precision</td>
</tr>
<tr>
<td>CAC</td>
<td>Combined accuracy and coverage</td>
<td>PA</td>
<td>Predictive ability</td>
</tr>
<tr>
<td>CATIE</td>
<td>Clinical antipsychotic trials of intervention effectiveness</td>
<td>PANSs</td>
<td>Positive and negative syndrome scale</td>
</tr>
<tr>
<td>CFR</td>
<td>Charging failure rate</td>
<td>PI</td>
<td>Popularity rate</td>
</tr>
<tr>
<td>CMU</td>
<td>Carnegie Mellon university</td>
<td>PO</td>
<td>Policy optimization</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural networks</td>
<td>POI</td>
<td>Point-of-interest</td>
</tr>
<tr>
<td>CR</td>
<td>Click reduction</td>
<td>POMDP</td>
<td>Partially observable Markov decision process</td>
</tr>
<tr>
<td>CTR</td>
<td>Click though rate</td>
<td>PP</td>
<td>Passenger pickup in 30 minutes</td>
</tr>
<tr>
<td>CVR</td>
<td>Conversion rate</td>
<td>PP30</td>
<td>Passenger pickup in 30 minutes</td>
</tr>
<tr>
<td>DDPG</td>
<td>Deep deterministic gradient descent</td>
<td>QE</td>
<td>Queries executed</td>
</tr>
<tr>
<td>DDQN</td>
<td>Double DQN</td>
<td>R</td>
<td>Recall</td>
</tr>
<tr>
<td>DHMM</td>
<td>Discriminative hidden Markov model</td>
<td>RA</td>
<td>Recommendation acceptance</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q network</td>
<td>RC</td>
<td>Recommendation score</td>
</tr>
<tr>
<td>DRL</td>
<td>Deep reinforcement learning</td>
<td>RDR</td>
<td>Recommended download rate</td>
</tr>
<tr>
<td>Earn</td>
<td>Average earning</td>
<td>RF</td>
<td>Reward formulation</td>
</tr>
<tr>
<td>ED</td>
<td>Exponential decay score</td>
<td>RL</td>
<td>Reinforcement learning</td>
</tr>
<tr>
<td>EMR</td>
<td>Entity matching rate</td>
<td>RLRS</td>
<td>Reinforcement learning based recommender system</td>
</tr>
<tr>
<td>ES</td>
<td>Energy saving</td>
<td>RMSE</td>
<td>Root-mean-square error</td>
</tr>
<tr>
<td>FC</td>
<td>Fully connected</td>
<td>RQ</td>
<td>Recommendation quality</td>
</tr>
<tr>
<td>FQI</td>
<td>Fitted Q iteration</td>
<td>RS</td>
<td>Recommender system</td>
</tr>
<tr>
<td>GANs</td>
<td>Generative adversarial networks</td>
<td>SAD</td>
<td>Session ad revenue</td>
</tr>
<tr>
<td>GI</td>
<td>Gini index</td>
<td>SDT</td>
<td>Session dwell time</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated recurrent units</td>
<td>SG</td>
<td>Shortcut gains</td>
</tr>
<tr>
<td>GVI</td>
<td>Gradient value iteration</td>
<td>SL</td>
<td>Session length</td>
</tr>
<tr>
<td>HR</td>
<td>Hit ratio</td>
<td>SR</td>
<td>State representation</td>
</tr>
<tr>
<td>HRL</td>
<td>Hierarchical reinforcement learning</td>
<td>ST</td>
<td>Session time</td>
</tr>
<tr>
<td>HSP</td>
<td>Historical song playlist</td>
<td>SuR</td>
<td>Success rate</td>
</tr>
<tr>
<td>ILS</td>
<td>Intra-list similarity</td>
<td>TSF</td>
<td>Total saving fee</td>
</tr>
<tr>
<td>k-NN</td>
<td>k-nearest neighbors</td>
<td>TPS</td>
<td>Taste profile subset</td>
</tr>
<tr>
<td>LFM</td>
<td>Last.fm</td>
<td>TWIS</td>
<td>Truncated weighted importance sampling</td>
</tr>
<tr>
<td>LRR</td>
<td>Low rank rate</td>
<td>UR</td>
<td>User rating</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long short-term memory</td>
<td>UV CTR</td>
<td>Click ratio to user view</td>
</tr>
<tr>
<td>LTR</td>
<td>Listening time ratio</td>
<td>VCT</td>
<td>Vacant cruising time</td>
</tr>
<tr>
<td>LTV</td>
<td>Life-time value</td>
<td>WI</td>
<td>Wage improvement</td>
</tr>
<tr>
<td>MARL</td>
<td>Multi-agent reinforcement learning</td>
<td>WPF</td>
<td>Weighted proportional fairness</td>
</tr>
<tr>
<td>MAP</td>
<td>Mean average precision</td>
<td>WQR</td>
<td>Wrong quite rate</td>
</tr>
<tr>
<td>MCP</td>
<td>Mean charging price</td>
<td>WT</td>
<td>Waiting time</td>
</tr>
<tr>
<td>MCCT</td>
<td>Monte Carlo tree search</td>
<td>YM</td>
<td>Yahoo music</td>
</tr>
<tr>
<td>MCWT</td>
<td>Mean charging wait time</td>
<td>YC</td>
<td>YooChoose</td>
</tr>
</tbody>
</table>
### 4.3 Reinforcement Learning based Recommender Systems Algorithms

In this section, we present algorithms in a classified manner. As discussed earlier, first we generally divide RLRSs into RL- and DRL-based methods. Then, we survey algorithms in each category with respect to the RLRS framework.

#### 4.3.1 RL-based RSs

In this section, we present RL-based RSs; i.e., methods that do not use deep learning for policy optimization. Table 4.2 provides a quick overview on RL-based methods.

**State Representation**

As illustrated in Table 4.2, apart from RPRMS and PHRR, all RL-based methods belong to either SR1 or SR2, which share almost the same proportion (see Fig. 4.7(a)). As mentioned earlier, methods in SR1 use all or a set/tuple of items for state representation. For example, WebWatcher [122], the first RLRS we identified, treats each item (i.e., web page) as a state in a web recommendation scenario. Similarly, Refs. [134] and [135] treat each author and...
learning object as a state in scientific collaborator recommendation and e-learning scenarios, respectively. As stated earlier, while this approach is possible in small state spaces, it is certainly not scalable when the item space grows large. Researchers found that keeping the track of a small set of items already rated/consumed by the user could be informative enough for policy optimization. Perhaps Refs. [19, 123] are the first RLRSs utilizing this idea, but the idea is better formalized for RLRSs by Ref. [1]. Specifically, in a web recommendation application, Taghipour and Kardan [1] borrow the N-gram model from the web usage mining literature [140] and introduce a sliding window to represent states, depicted in Fig. 4.6. In this figure, circles are states, right arrows are actions, and $V$ and $R$ indicate visited and previously recommended pages, respectively. While using this model, the authors assume that knowing the last $k$ pages visited by the user provides enough information to predict their future page requests. It is noteworthy to mention that this set or sliding window in SR1 could indicate any useful information for the purpose of policy optimization, including a set of commercial items [19], concepts in a website [123, 125], emotion classes of songs [127], skills [139], and music songs [94]. In a different setting, Choi et al. [90] formulate the recommendation problem as a gridworld game and each grid cell, with its users and items inside, is considered as a state.

Other researchers have proposed to extract some features from user, items, and context and use them for state representation (SR2). Among the first attempts in SR2 is Mahmood et al. works [124, 126, 130] in which a set of variables from user (e.g., the number of times the user has modified his query), agent (e.g., previous action of the agent), and interaction session (e.g., the number of episodes elapsed) are used for state representation. A similar approach
Figure 4.7: The summary of four components of RLRS framework in RL-based methods is used in RLRadio [93] where some variables, containing information about radio channels of user interest and their listening behavior, are defined to represent states. DJ-MC [131] uses an encoding method to represent each song as a vector of song descriptors and each state is the concatenation of $k$ songs vectors in the playlist. Features from weather conditions and time are used in CAPR [137] for state representation in a point of interest (POI) recommender.

SR2 is specifically popular in healthcare applications in which information about patients are typically recorded by several descriptive features [128, 129]. In a different setting, POMDP-Rec [133] formulates states as belief states using low-dimensional factor model [141]. More precisely, with a partially observed user-item matrix, user’s behavior observations (O), items’ latent features (V), and users’ latent interests (U) can be calculated as

$$p(O|U, V, \sigma^2) = \prod_{i=1}^{n} \prod_{j=1}^{m} \mathcal{N}(O_{ij}|U_i^\top V_j, \sigma^2)^{I_{ij}},$$

(4.22)
\begin{align}
  p(U|\sigma^2_U) &= \prod_{i=1}^N \mathcal{N}(U_i|0, \sigma^2_U I), \\
  p(V|\sigma^2_V) &= \prod_{j=1}^N \mathcal{N}(V_j|0, \sigma^2_V I),
\end{align}

where \( \mathcal{N}(x|\mu, \sigma^2) \) is the probability density function of Gaussian distribution with mean \( \mu \) and variance \( \sigma^2 \), \( I_{ij} \in \{0, 1\} \), and \( I_{ij} = 1 \) means that user \( i \) has rated item \( j \). A belief state is then a concatenation of \( U \) and \( V \), i.e., \( b_{ij} = (U_i, V_j) \).

The only works in RL-based methods that lie in SR3 are RPRMS [136] and PHRR [138], both are in music recommendation domain. In RPRMS, states are a concatenation of songs lyrics embeddings, generated using Word2Vec [142], and audio embeddings, generated using a pre-trained WaveNet model [143]. PHRR uses weighted matrix factorization [144] and CNN to embed songs, and similar to DJ-MC, each state is the concatenation of several song vectors.

**Policy Optimization**

According to Fig. 4.7(d), temporal difference methods, i.e., Q-learning and Sarsa, have been the most popular RL algorithms among RL-based methods [122, 1, 125, 127, 130, 94, 90, 135, 136, 139]. The main reason of this popularity is their simplicity; that is, they are online, model-free, need minimal amount of computation, and can be expressed by a single equation (see Eqs. (4.4) and (4.5)) [25]. Applying Q-learning/Sarsa for policy optimization is quite straightforward and need not any specific modification. Researchers in [1] use a simple trick to have a decreasing learning rate \( \alpha = 1/1+\text{visits}(s, a) \) in Eq. (4.4), which helps algorithm convergence. This trick is also used in [127, 130, 94]. A problem with temporal difference methods, like any tabular RL method, is that they lead to the curse of dimensionality [145]. To tackle this problem, as discussed earlier, researchers try to manage the state space and keep it small enough.

Among the tabular methods, dynamic programming methods are usually impractical due to their great computational expense and the need to perfect knowledge about the environ-
ment. While these algorithms are polynomial in the number of states, performing even one iteration of policy or value iteration methods is often infeasible [146]. To make it practical, Ref. [19] uses a couple of features in their state space and makes some approximations. For instance, one feature of state space in [19] is directionality; the authors argue that a short state cannot follow a long state or the probability of occurring loops in their MDP is not very high. Moreover, Ref. [124] keeps the number of policy iteration run to a limited number.

In contrast to dynamic programming, Monte Carlo methods do not need a perfect knowledge (or model) of the environment. Instead, they only need sampled experience, i.e., some interactions with the environment. However, Monte Carlo methods do not bootstrap, and they update the value function only after a complete episode, which makes it their convergence slow. MCTS is a decision-time planning algorithm that benefits from online, incremental, sample-based value estimation and policy improvement [25] and has been employed by [131, 137, 138]. In order to facilitate the agent learning, in case the song space is very large or search time is limited, DJ-MC [131] clusters songs according to song types and then applies MCTS to clustered songs. PHRR [138] adopts similar schemes in policy optimization and song clustering. Similar to AlphaGo [102], CAPR [137] uses UCT (Upper Confidence Bound applied to Trees) [147] to solve the exploration vs exploitation trade-off in the selection step of MCTS (see section 4.2.2).

Preda and Popescu [123] use Sarsa(\(\lambda\)) with *tile coding* [25] linear approximation for policy optimization. To be able to apply Sarsa(\(\lambda\)), the work transforms epistemic information into arrays of real numbers. Moling et al. [93] define the problem of optimal radio channel recommendation as a continuous task and then employ R-learning [105] to solve it.

On the other hand, some RL-based RSs have used approximate methods for policy optimization, including fitted Q [129, 128, 132, 133] and gradient value iteration [134]. Fitted Q is a flexible framework that can fit any approximation architecture to Q-function [107]. Accordingly, any batch-mode supervised regression algorithms can be used to approximate the Q-function, which can scale well to high dimensional spaces [25]. However, one problem
with this method is that it could have a high computational and memory overhead with the increase in the number of four-tuples \((x_t, u_t, r_t, x_{t+1})\), where \(x_t\) indicates the system state at time \(t\), \(u_t\) the control action taken, \(r_t\) the immediate reward, and \(x_{t+1}\) the next state of the system [107]. This algorithm has been used by several RLRSs [129, 128, 132, 133]. To fit the Q function in these methods, linear regression [128, 132], support vector regression [129], and neural networks [133] are used. Finally, Zhang et al. [134] introduce a gradient descent version of value iteration algorithm for collaborator recommendation in a multi-agent RL setting.

**Reward Formulation**

Fig. 4.7(b) shows that R2, with 60% proportion, has been more popular than R1, with 40%, among RL-based methods. In R1, different numerical values have been used for the immediate reward. For instance, Mahmood et al. use +1 in terminal state and a negative number otherwise [124], +5 for adding a product to travel plan, +1 for showing result page, 0 otherwise [126], and +100 for buying a book, -30 for user quit, and 0 otherwise [130]. On the other hand in R2, researchers have proposed to use different observations from the environment to formulate the reward, including net profit [19], overall survival time [129], some clinical scores (i.e., PANSS) [128], and Jaccard distance between two states [90].

**Environment Building**

It is observable from Fig. 4.7(c) that the dominant environment building method in RL-based RSs is offline. This makes sense as training the agent and testing the performance on an available dataset is the easiest and safest option. Two popular datasets used in RL-based RSs are MovieLens [148] and Million Song dataset [149].

Another environment building method is simulation, which is a safe method to fine-tune important model parameters before system deployment or an online study. Simulation could be as simple as assuming constant users with predefined preference patterns [127, 129], or it
could be more complex and learn user behavior through available data [124, 131, 130]. For example, in a supervised learning task, Mahmood et al. [124] define a user behavior model and use it in a travel RS, called NutKing, to learn transition probabilities. The aim is to know how the simulated user reacts to a certain action of the system.

Online study is the most effective, but costly environment building method for RLRSs. In an early, valuable attempt [19], the performance of the proposed MDP-based RS is evaluated in a two-year online study conducted on an online book store. The study has had a good daily exposure to users, almost 5000-6000 different users daily, with a reasonable number of items to recommend (over 15,000), compared to other online studies performed by RL-based methods with only five [127], 13 [130], 47 [131], 469 [126], and 500 [123] users.

### 4.3.2 DRL-based RSs

In this section, we study DRL-based methods; those RSs that use a deep learning model for policy optimization. Table 4.3 provides a quick overview of these methods.

**State Representation**

As depicted in Fig. 4.8(a), SR3 is the dominant state representation scheme for DRL-based RSs. As stated earlier, this is because deep models are trained more effectively on dense, low-dimensional vectors. Nonetheless, researchers have taken one step further and tried to make the general framework of SR3 (see Fig. 4.5(c)) more effective. Typically, in RLRSs, items positively rated by the user are considered as the preferences of the user. However, in DEERS [2], authors discuss that the proportion of negative feedback, e.g., skipped items, could be much larger than the positive one, so they propose to have two states: positive and negative states. Fig. 4.9 illustrates this modification. In particular, the input is divided into items with positive and negative feedbacks, are passed through embedding and RNN layers, and fed into Q network where they are concatenated. This technique has also inspired other researchers [158, 200]. Instead of RNN layer, DRCGR [158] uses a convolution layer (with both horizontal and vertical kernels) to encode the embeddings of positive feedbacks. On the
<table>
<thead>
<tr>
<th>RLBS</th>
<th>Year</th>
<th>SM</th>
<th>DB</th>
<th>RF</th>
<th>Metrics</th>
<th>Dataset</th>
<th>Application</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slate-MDP</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Simulation</td>
<td>Reward</td>
<td>N/A</td>
<td>Healthcare</td>
<td></td>
</tr>
<tr>
<td>Wolpertinger</td>
<td>2015</td>
<td>SR1</td>
<td>DDPG</td>
<td>Simulation</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neudel et al.</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Simulation</td>
<td>Reward</td>
<td>N/A</td>
<td>MMIC</td>
<td></td>
</tr>
<tr>
<td>CEI</td>
<td>2015</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Simulation</td>
<td>P, F, BLEU, Reward</td>
<td>ML1M, MT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iffin et al.</td>
<td>2015</td>
<td>SR2</td>
<td>Dueling DQN</td>
<td>Offline</td>
<td>Mortality Rate</td>
<td>MMIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deni [5]</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline, Simulation</td>
<td>MAP, NDCG</td>
<td>Taobao, E-commerce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robust DQN</td>
<td>2015</td>
<td>SR2</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRL-RNN</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEM [5]</td>
<td>2015</td>
<td>SR1</td>
<td>Dueling DQN</td>
<td>Offline, Online</td>
<td>CTR, NDCG, P, R, S</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Page</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline, Simulation</td>
<td>P, R, F, NDCG, MAP</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Mnih et al.</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CapDRL</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K, ML1M</td>
<td>Movie</td>
<td></td>
</tr>
<tr>
<td>Proddue [157]</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Offline, Simulation</td>
<td>ACS, ADS, ART</td>
<td>N/A</td>
<td>Feed streaming</td>
<td></td>
</tr>
<tr>
<td>DQoG</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIRD</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Simulation</td>
<td>MAP, NDCG</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Slate-Q</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Simulation, Online</td>
<td>Reward, A, Q</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vu et al.</td>
<td>2015</td>
<td>SR1</td>
<td>Dueling</td>
<td>Simulation</td>
<td>S, N, NY, HP</td>
<td>UT-Zappos50K</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Tomasini</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhang et al.</td>
<td>2015</td>
<td>SR2</td>
<td>REINFORCE-wb</td>
<td>Offline</td>
<td>Reward, P, R, F, I</td>
<td>ML1M, Netflix</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>DRESS</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liu et al.</td>
<td>2015</td>
<td>SR1</td>
<td>Dueling DQN</td>
<td>Offline</td>
<td>Simulation</td>
<td>AAR</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Yuan et al.</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K, ML1M</td>
<td>Movie</td>
<td></td>
</tr>
<tr>
<td>CROMA</td>
<td>2015</td>
<td>SR2</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>Twitter Tweet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REINFORCE</td>
<td>2015</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Simulation</td>
<td>ViewTime</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFR [160]</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K, ML1M, YM</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>UQDN</td>
<td>2015</td>
<td>SR2</td>
<td>DQN</td>
<td>Offline, Simulation</td>
<td>ACS, ADS, ART</td>
<td>N/A</td>
<td>Feed streaming</td>
<td></td>
</tr>
<tr>
<td>RoGAN</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>P, R, HD, NDCG</td>
<td>Amazon</td>
<td>E-commerce</td>
</tr>
<tr>
<td>TPGIR [179]</td>
<td>2015</td>
<td>SR1</td>
<td>DUQN</td>
<td>Offline</td>
<td>Reward, P, R, F</td>
<td>ML100M</td>
<td>Netflix</td>
<td></td>
</tr>
<tr>
<td>Do-RMFTS [174]</td>
<td>2015</td>
<td>SR1</td>
<td>DUQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Cascading DQN</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML, LFM, Yp, YC, APN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EIR [175]</td>
<td>2015</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Offline</td>
<td>Reward, P, R, HD, NDCG</td>
<td>Amazon</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Paul et al.</td>
<td>2015</td>
<td>SR1</td>
<td>DUQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>reEnergy</td>
<td>2015</td>
<td>SR2</td>
<td>DUQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>Energy optimization</td>
<td></td>
</tr>
<tr>
<td>GCQX [179]</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, LFM, Pinterest</td>
<td>ML100K, Xina</td>
<td>E-commerce</td>
</tr>
<tr>
<td>MEDAR</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline, Simulation</td>
<td>MAP, NDCG, Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>DHR [3, 4]</td>
<td>2015</td>
<td>SR3</td>
<td>DDQN</td>
<td>Offline, Simulation</td>
<td>MAP, NDCG, Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>FairRec</td>
<td>2015</td>
<td>SR2</td>
<td>DUQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, YM, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAR [182]</td>
<td>2015</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Simulation</td>
<td>Reward</td>
<td>ML100K</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>MASSA [183]</td>
<td>2015</td>
<td>SR1</td>
<td>DUQN</td>
<td>Offline, Simulation</td>
<td>P, NDCG</td>
<td>Taobao</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>DeepChase [184]</td>
<td>2015</td>
<td>SR1</td>
<td>DUQN</td>
<td>Offline</td>
<td>Simulation</td>
<td>MAP, NDCG, Reward</td>
<td>JD</td>
<td>E-commerce</td>
</tr>
<tr>
<td>KGPolicy</td>
<td>2015</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Offline</td>
<td>Reward</td>
<td>N, R, NDCG</td>
<td>Amazon, LFM, Yelp</td>
<td>E-commerce</td>
</tr>
<tr>
<td>KGL [186]</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>CRP [188]</td>
<td>2015</td>
<td>SR1</td>
<td>DQN</td>
<td>Simulation</td>
<td>S, N, NY, HP</td>
<td>UT-Zappos50K</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Xin et al.</td>
<td>2015</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>ADAC [196]</td>
<td>2015</td>
<td>SR1</td>
<td>Act-Critic</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>KRL [190]</td>
<td>2016</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Offline</td>
<td>Reward</td>
<td>N, R, NDCG</td>
<td>Amazon, LFM</td>
<td>E-commerce</td>
</tr>
<tr>
<td>DRekoning</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML, LFM</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>KGQR [199]</td>
<td>2016</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Singh et al.</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Simulation</td>
<td>Reward, Risk</td>
<td>ML1M</td>
<td>E-commerce</td>
</tr>
<tr>
<td>EDR [196]</td>
<td>2016</td>
<td>SR1</td>
<td>DDQN</td>
<td>Offline</td>
<td>Simulation</td>
<td>P, NDCG, MAP, Reward</td>
<td>ML1M, Jester</td>
<td></td>
</tr>
<tr>
<td>SBH [196]</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Simulation</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPRL [197]</td>
<td>2016</td>
<td>SR1</td>
<td>DDPG</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DQCh [198]</td>
<td>2016</td>
<td>SR1</td>
<td>DDPG</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCPO [199]</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCB [200]</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DHR [201]</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M, ML100K, HC, Jester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASTER [203]</td>
<td>2016</td>
<td>SR2</td>
<td>DDQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N, R, NDCG, HH</td>
<td>Amazon, LFM, Yelp</td>
<td>E-commerce</td>
</tr>
<tr>
<td>Anachkoo [204]</td>
<td>2016</td>
<td>SR1</td>
<td>Act-Critic</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>UNICORN</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>HRL-Rec</td>
<td>2016</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>Peng et al.</td>
<td>2017</td>
<td>SR2</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>ANR [207]</td>
<td>2017</td>
<td>SR2</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>DEAR [208]</td>
<td>2017</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>VPP [209]</td>
<td>2017</td>
<td>SR1</td>
<td>DQN</td>
<td>Offline</td>
<td>Reward</td>
<td>N/A</td>
<td>E-commerce</td>
<td></td>
</tr>
<tr>
<td>SMART-Rec</td>
<td>2017</td>
<td>SR1</td>
<td>Act-Critic</td>
<td>Offline</td>
<td>Reward</td>
<td>ML100K, ICRM, Movie</td>
<td></td>
<td></td>
</tr>
<tr>
<td>URL [211]</td>
<td>2017</td>
<td>SR1</td>
<td>REINFORCE-wb</td>
<td>Offline</td>
<td>Reward</td>
<td>ML1M</td>
<td>E-commerce</td>
<td></td>
</tr>
</tbody>
</table>
other hand, a generative adversarial network (GAN) module is trained to generate negative samples. Deep Page [92] also extends the SR3 framework by adding a CNN module between the embedding and RNN layers, in order to learn item spatial display scheme in a page-wise recommendation scenario. Before passing the item embeddings through the CNN module, a page layer is used to convert item embeddings into a 2D grid/matrix for 2D CNN processing. Moreover, authors in [172] propose to use a position weighting scheme for state embedding. Formally, if $W$ is a matrix with historical steps as rows and importance weight of positions as columns, the embedding of a state $s^t$ can be defined as

$$s^t = h(F^{t-m:t-1}) = vec[\sigma(F^{t-m:t-1}W + B)],$$

(4.25)

where $F$ is the feature vector of the history with $m$ steps, $B$ is a bias matrix, $\sigma(\cdot)$ is a nonlinear activation, and $vec[\cdot]$ concatenates the matrix columns. The authors claim that
this method for state embedding is more efficient for optimization than LSTM. Finally, in D^2RLIR [197], a positional encoding is added to state embeddings so that the model understands the chronological order of items.

In DRR [3], an individual module called state representation module is proposed for the purpose of state formulation. Authors propose three structures to model the interactions between user and items. The first structure, DRR-p, simply concatenates the embeddings of items and their pairwise products, as depicted in Fig. 4.11(a). More formally, if \( H = \{v_1, v_2, ..., v_n\} \) is the positive interaction history of the user and

\[
P = \{w_i v_i \otimes w_j v_j | i, j = 1, 2, ..., n\}
\]  

(4.26)

is the weighted pairwise product between items, then state \( S \) is defined as the concatenation
of $H$ and $P$, i.e., $S = (H, P)$. In the second structure, DRR-u, the user embedding is also incorporated (shown in Fig. 4.11(b)). That means, with

$$K = \{u \otimes w_i v_i | i = 1, 2, ..., n\}, \quad (4.27)$$

$S = (K, P)$. In the last structure illustrated in Fig. 4.11(c), DRR-ave, an average pooling layer is introduced to eliminate the items’ position bias in the recommended list. In particular, if

$$G = \{\text{ave}(w_i v_i) | i = 1, ..., n\}, \quad (4.28)$$

$S = (u, u \otimes G, G)$. In [4], the authors extend DRR-ave and add an attention network to generate user-dependent weights for each item, as depicted in Fig. 4.11(d). In another work [195], the same authors study the effect of updating the state representation module using a supervised learning signal, and through experimental studies, they show that the recommendation performance could be improved.

Around 20% of DRL-based RSs belong to SR2. For instance, DRN [5] uses user and
context features for the purpose of state representation in news recommendation. User features extracted in DRN include the features of the news clicked by the user in different time frames, like one hour, six hours, 24 hours, one week, and one year. These news features include headline provider, ranking, entity name, category, and topic category. Context features used also describe the time context of the news request, including time and weekday. Similar to RL-based methods, SR2 is the popular state representation method in healthcare applications [151, 153]. Nemati et al. [151] use a partially-observable MDP (POMDP) formulation in a clinical application. They formulate states as belief states using discriminative hidden Markov model (DHMM).

We found only two works [7, 150] lie in SR1. Perhaps the reason these works use a simple state representation method is that they are representative works not specifically designed for RSs, but developed to target specific challenges in applying DRL to domains like RSs.

Policy Optimization

After defining states, the role of policy $\pi$ is to map states to actions. Policy optimization algorithms used by DRL-based RSs could be generally divided into value-based, policy gradient, and actor-critic methods.

**Value-based methods.** Apart from MCTS used in [171], DQN and its extensions, i.e., DDQN, dueling DQN, and dueling DDQN, are the ruling value-based methods. Basically, there are three main elements in DQN: 1) Q network architecture, 2) experience replay, and 3) exploration. We survey DQN-based methods according to these elements and Table 4.4 summarizes DQN-based methods.

1) **Q network Architecture.** Fig. 4.10 depicts two possible architectures of Q network used by DQN-based RLRSs. The original architecture (A1), introduced in [110, 111], receives the state and emits the Q value of all actions, indicated by $Q_1, \ldots, Q_n$ in Fig. 4.10(a). While A1 works fine when the action space is small, its applicability to the RS domain with a large, and even huge (in the order of millions), action space is questionable. Another possible
Table 4.4: DQN-based RSs

<table>
<thead>
<tr>
<th>RLRS</th>
<th>Year</th>
<th>Algorithm</th>
<th>Architecture</th>
<th>Experience replay</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slate-MDP [150]</td>
<td>2015</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>Nemati et al. [151]</td>
<td>2016</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform*</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>Raghu et al. [153]</td>
<td>2017</td>
<td>Dueling DDQN</td>
<td>A1</td>
<td>Prioritized</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>DEERS [2]</td>
<td>2018</td>
<td>DQN</td>
<td>A2</td>
<td>Prioritized</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>Robust DQN [89]</td>
<td>2018</td>
<td>DDQN</td>
<td>A2</td>
<td>Stratified</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>FeedRec [157]</td>
<td>2019</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>Decayed ε-greedy</td>
</tr>
<tr>
<td>DRCGR [158]</td>
<td>2019</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>SlateQ [120]</td>
<td>2019</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>Tsumita [160]</td>
<td>2019</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform*</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>Liu et al. [163]</td>
<td>2019</td>
<td>Dueling DDQN</td>
<td>A1</td>
<td>Uniform</td>
<td>Decayed ε-greedy</td>
</tr>
<tr>
<td>UdQN [167]</td>
<td>2019</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>Cascading DQN [172]</td>
<td>2019</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>Pseudo Dyna-Q [174]</td>
<td>2020</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>Decayed ε-greedy</td>
</tr>
<tr>
<td>Zhao et al. [176]</td>
<td>2020</td>
<td>Duleing DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>SADQN [175]</td>
<td>2020</td>
<td>DQN</td>
<td>A2</td>
<td>Not using</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>CPR [188]</td>
<td>2020</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>Xin et al. [189]</td>
<td>2020</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>KGQR [193]</td>
<td>2020</td>
<td>Dueling DDQN</td>
<td>A2</td>
<td>Uniform</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>EDRR [195]</td>
<td>2020</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform*</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>GCQN [179]</td>
<td>2020</td>
<td>DQN</td>
<td>A2</td>
<td>Uniform*</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>SRR [196]</td>
<td>2020</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform*</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>DICRS [200]</td>
<td>2021</td>
<td>DQN</td>
<td>A1</td>
<td>Uniform</td>
<td>ε-greedy*</td>
</tr>
<tr>
<td>UNICORN [205]</td>
<td>2021</td>
<td>Dueling DDQN</td>
<td>A2</td>
<td>Prioritized</td>
<td>ε-greedy</td>
</tr>
<tr>
<td>GoalRec [207]</td>
<td>2021</td>
<td>DQN</td>
<td>A2</td>
<td>Hindsight</td>
<td>Decayed ε-greedy</td>
</tr>
<tr>
<td>DEAR [208]</td>
<td>2021</td>
<td>Dueling DQN</td>
<td>Hybrid</td>
<td>Uniform</td>
<td>ε-greedy*</td>
</tr>
</tbody>
</table>

* There is no indication about this element in respective papers and we have assumed uniform/ε-greedy because all these algorithms are based on DQN.

architecture (A2) is to receive the pair of state and action, and then to emit the Q value of the pair, i.e., \(Q(s, a)\) (depicted in Fig. 4.10(b)). Although A2 solves A1’s problem, a problem with A2 is that the time complexity of the model could be high.

Despite the original DQN where CNN is used for Q network to process the image data, Q network in RLRSSs is typically composed of several FC layers, as the input, i.e., states or actions, are in the form of 1D vectors. For example, as stated before, DEERS [2] uses two types of states as the input into Q network: positive and negative states, depicted in 4.9. Q network is a five-layer FC network where the first three layers are separate for positive and negative states, and then the last two layers connect both states, emitting the Q value of a given state and action pair. To take this dual-state architecture into account, the original loss function of DQN in Eq. (4.14) is modified as

\[
L(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s_+, s_-, a; \theta_i))^2 \right],
\]  

(4.29)
where \( y_i = \mathbb{E}_{s'}[r + \gamma \max_a Q(s'_+, s'_-, a'; \theta_{i-1}) | s_+, s_-, a] \). Consequently, the gradient of the loss function becomes

\[
\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (r + \gamma \max_a Q(s'_+, s'_-, a'; \theta_{i-1}) - Q(s_+, s_-, a; \theta_i)) \nabla_{\theta_i} Q(s_+, s_-, a; \theta_i) \right].
\]

Other researchers have employed DQN’s extensions. For instance, DRN [5] adopts dueling DDQN for policy optimization in news recommendation. In particular, the authors argue that while the reward of taking an action is impacted by all features, i.e., user, news, context, and user-news features, there is a reward that is impacted by merely user and context features. Accordingly, the Q function is divided into value function \( V(s) \) and advantage function \( A(s, a) \). As depicted in Fig. 4.12, while \( V(s) \) is fed with state features, the input into \( A(s, a) \) is comprised of state and action features.

DEAR [208] studies the problem of advertising along with recommendation. It combines the two architectures of DQN Q network, i.e., A1 and A2, and the resulting architecture generates the Q value of a list of candidate ads if inserted in the recommendation list. In other words, the input is similar to A2 architecture, i.e., state and action, and the output is the same as A1, which is a list containing the Q values of all state-action pairs.

2) Experience Replay. According to Table 4.4, the vast majority (22 out of 28) of DQN-based RSs use the original uniform sampling to replay collected experiences. Also, only
three of them use prioritized experience replay \cite{153, 2, 205}. Authors in \cite{89} propose to use *stratified sampling* replay instead of uniform sampling to address the variance of sampling in dynamic environments. Stratified sampling is a sampling technique from a population in which the entire population is partitioned into several groups (called strata) and then samples are randomly selected from these strata \cite{212}. They propose to use some stable features from customers, like gender, age, and geography, as strata.

GoalRec \cite{207} uses *hindsight* experience replay \cite{213}. The main idea in hindsight replay is to learn from an undesirable outcome as much as from a desirable outcome. Since the goal has no effect on the dynamics of the environment, a failed trajectory is re-labelled as a successful one, as if the state in the trajectory is the actual goal. This considerably improves sample efficiency.

In contrast to existing DQN-based methods, SADQN \cite{175} does not use experience replay for training. Instead, in each episode of the training phase, a user is sampled from the user set and the agent is trained on the available interactions until it is converged. Using experiments, authors claim that the experience replay in fact diminishes the performance of SADQN.

3) **Exploration.** Although exploration is an important factor in learning of the agent, many DQN-based methods have seemingly overlooked it, as there is no specific indication about this in respective publications. Apart from simple exploration techniques like $\epsilon$-greedy, DRN \cite{5} proposes to use an exploration approach similar to *dueling bandit gradient descent* algorithm \cite{214}. In particular, there is a separate network for exploration called *explore network* and its parameters can be obtained using a disturbance to the parameters of current network with parameters $W$

$$\Delta W = \alpha \cdot \text{rand}(-1, 1) \cdot W, \quad (4.31)$$

where $\alpha$ is the explore coefficient. Then, the agent generates a merged list of recommendations using probabilistically interleaving between items found by current network and explore
In recEnergy [178], to balance the exploration vs exploitation trade-off, Boltzmann exploration [25] is used. More precisely, the output Q values of actions from Q network are passed through a softmax equation as

\[ P(a) = \frac{\exp \frac{Q(a)}{\tau}}{\sum_{i=1}^{n} \exp \frac{Q(i)}{\tau}}, \]

where \( \tau \) is a temperature and is decayed over time. This method guarantees that the model explores more often initially, and then it starts to exploit actions with larger Q values more frequently.

**Policy Gradient methods.** In contrast with value-based methods, policy gradient methods learn a parameterized policy without the need to a value function. REINFORCE is a Monte Carlo, stochastic gradient method that directly updates the policy weights. The major problems of REINFORCE algorithm are high variance and slow learning. These problems come from the Monte Carlo nature of REINFORCE, as it selects samples randomly and updates are made when the episode is completed.

In a valuable work, Ref. [6] adapts REINFORCE algorithm to a neural candidate generator with a very large action space. In particular, in an online RL setting, the estimator of the policy gradient can be expressed as

\[ \sum_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^{\tau} R_t \Delta_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \]

where \( \pi_{\theta} \) is the parametrized policy, \( \tau = (s_0, a_0, s_1, ...) \), and \( R_t \) is the cumulative reward. Since in the RS setting, unlike classical RL problems, the online or real time interaction between the agent and environment is infeasible and usually only logged feedback is available, applying the policy gradient in Eq. (4.33) is biased and needs correction. The off-policy-
corrected policy gradient estimator is then:

$$
\sum_{\tau} \frac{\pi_\theta(\tau)}{\beta(\tau)} \left[ \sum_{t=0}^{|\tau|} R_t \Delta_\theta \log \pi_\theta(a_t|s_t) \right],
$$

where $\beta$ is the behavior policy and

$$
\frac{\pi_\theta(\tau)}{\beta(\tau)} = \frac{\rho(s_0) \prod_{t=0}^{|\tau|} P(s_{t+1}|s_t, a_t) \pi(a_t|s_t)}{\rho(s_0) \prod_{t=0}^{|\tau|} P(s_{t+1}|s_t, a_t) \beta(a_t|s_t)} = \prod_{t=0}^{|\tau|} \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)}
$$

is the importance weight. Since this correction generates a huge variance for the estimator due to the chained products, authors use first-order approximation for importance weights, leading to the following biased estimator with a lower variance for the estimator:

$$
\sum_{\tau} \left[ \sum_{t=0}^{|\tau|} \frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} R_t \Delta_\theta \log \pi_\theta(a_t|s_t) \right].
$$

Fig. 4.13 illustrates the neural architecture of the parametrized policy $\pi_\theta$ in Eq. (4.36).

As discussed in section 4.2.2, REINFORCE-wb adds a baseline to REINFORCE’s update rule in order to decrease the variance (see Eq. (4.12)). Several RLRSs have used this approach [152, 166, 8, 185, 202]. Specifically, the baseline in these methods is a value net-
work [152, 8, 202], a constraint [166], and average reward [185]. However, it is not clear how other REINFORCE-based RSs [155, 161, 173, 177, 182, 192, 194, 201] tackle the variance problem.

Following SeqGAN [215], IRecGAN [168] employs GANs to develop a model-based RL recommender. In particular, the generator is responsible to generate recommendations and to model user behavior, and the discriminator is used to rescale the generated rewards. Using both generated and offline data, REINFORCE is used to optimize the recommendation policy. Similar to SeqGAN, to reduce the variance, IRecGAN uses MCTS with roll-out policy, i.e., sampling $N$ sequences from interaction between the recommender and user model and then averaging the estimations.

**Actor-Critique Methods.** DDPG is the base method used in almost all actor-critic based RLRSs. DDPG uses an actor-critic architecture to combine DPG and DQN. Actor, also called policy network, is responsible to generate actions, and critic, a DQN module, is responsible to evaluate the action taken. The original DDPG uses either several FC layers or convolutional plus FC layers when the input is pixel. The output layer of actor is a $\text{tanh}$ layer to bound actions. For exploration, DDPG uses a temporally correlated noise, Ornstein-Uhlenbeck (OU) process [216], that is suitable for physical environments with momentum. Also, similar to DQN, experience replay with uniform sampling is used.

Wolpertinger [7] is the first actor-critic method based on DDPG to handle large discrete action spaces, with a recommendation case study. The idea is to provide a method that has sub-linear complexity w.r.t. action space and generalizable over actions. As depicted in Fig. 4.14, Wolpertinger consists of two parts: action generation and action refinement. In the first part, proto-actions are generated by the actor in continuous space and then are mapped to discrete space using $k$-nearest neighbor ($k$-NN) method. More precisely, the proto-action $\hat{a}$ is generated by actor as

$$\hat{a} = f_\theta(s). \quad (4.37)$$
This proto-action is not likely to be a valid action so $\hat{a}$ is mapped to an element in $\mathcal{A}$ as

$$g_k(\hat{a}) = \arg \min_{a \in \mathcal{A}} |a - \hat{a}|_2. \quad (4.38)$$

In the second part, outlier actions are filtered using a critic, which selects the best action that has the maximum $Q$ value. In other words,

$$\pi_\theta(s) = \arg \max_{a \in g_k} Q_\theta(s, a). \quad (4.39)$$

Wolpertinger is trained using DDPG. For exploration, for the recommendation task, Wolpertinger uses a guided $\epsilon$-greedy exploration technique. In particular, the exploration is restricted to a likely good set of actions provided by the environment simulator.

The vast majority of actor-critic methods are based on DDPG [7, 154, 92, 3, 91, 156, 86, 165, 180, 4, 181, 184, 186, 195, 196, 197, 198, 183, 199, 206, 203]. Table 4.5 summarizes these methods. As depicted, only DRR uses prioritized experience replay; the remaining algorithms either use uniform sampling or there is no clue about this in respective publications. Another worthwhile observation from Table 4.5 is that the vast majority of algorithms do not talk about exploration. Of five algorithms with a described exploration method, three of them,
### Table 4.5: DDPG-based RSs

<table>
<thead>
<tr>
<th>RLRS</th>
<th>Year</th>
<th>Experience replay</th>
<th>Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wolpertinger</td>
<td>2015</td>
<td>Uniform</td>
<td>Guided $\epsilon$-greedy</td>
</tr>
<tr>
<td>SRL-RNN [154]</td>
<td>2018</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>Deep Page [92]</td>
<td>2018</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>DRR [3, 4]</td>
<td>2018, 2020</td>
<td>Prioritized</td>
<td>$\epsilon$-greedy</td>
</tr>
<tr>
<td>Munemasa et al. [91]</td>
<td>2018</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>CapDRL [156]</td>
<td>2019</td>
<td>Uniform*</td>
<td>N/A</td>
</tr>
<tr>
<td>LIRD [86]</td>
<td>2019</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>CROMA [165]</td>
<td>2019</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>MaHRL [180]</td>
<td>2020</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>FairRec [181]</td>
<td>2020</td>
<td>Uniform*</td>
<td>N/A</td>
</tr>
<tr>
<td>DeepChain [184]</td>
<td>2020</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>KGRL [186]</td>
<td>2020</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>EDRR [195]</td>
<td>2020</td>
<td>Uniform*</td>
<td>N/A</td>
</tr>
<tr>
<td>SRR [196]</td>
<td>2020</td>
<td>Uniform*</td>
<td>N/A</td>
</tr>
<tr>
<td>MASSA [183]</td>
<td>2021</td>
<td>Uniform</td>
<td>Entropy-regularized</td>
</tr>
<tr>
<td>FCPO [199]</td>
<td>2021</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>HRL-Rec [206]</td>
<td>2021</td>
<td>Uniform*</td>
<td>$\epsilon$-greedy</td>
</tr>
<tr>
<td>MASTER [203]</td>
<td>2021</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>D²RLIR [197]</td>
<td>2021</td>
<td>Uniform</td>
<td>N/A</td>
</tr>
<tr>
<td>DRGR [198]</td>
<td>2021</td>
<td>Uniform*</td>
<td>OU noise</td>
</tr>
</tbody>
</table>

* There is no indication about this element in respective papers and we have assumed uniform because all these algorithms are based on DDPG.

i.e., Wolpertinger, DRR, and HRL-Recused, are based on $\epsilon$-greedy. Similar to DDPG, DRGR uses OU process to encourage better exploration for the actor. However, as stated earlier, OU noise is suitable for physical processes. Finally, MASSA introduces a novel entropy-regularized method for exploration, a method similar to soft actor-critic [217].

Actor-critic seems as a popular architecture for multi-agent RL (MARL). Centralized learning/training with decentralized execution [218, 219] is a suitable framework for a multi-agent setting and adopted by CROMA, MASSA, DeepChain, and MASTER. For instance, Fig. 4.15(a) depicts the architecture of MADDPG [219], which utilizes a centralized training and decentralized execution framework. MASSA builds upon this architecture and adds a signal network to the MADDPG architecture, depicted in Fig. 4.15(b), which is responsible to ease the cooperation between decentralized actors.

There are a couple of actor-critic based methods that use adversarial training for a better policy learning [187, 190]. For instance, CRSAL [187] extends soft actor-critic [217] with adversarial learning, by adding a discriminator inside the critic to distinguish between dialogues generated by the policy network and real users. In a path reasoning scenario over knowledge graph, ADAC [190] uses adversarial imitation learning [220] and defines two path
and meta-path discriminators to distinguish expert paths from paths generated by the actor.

In contrast to other actor-critic methods, DRESS [162] uses PPO and SDAC [210] proposes a stochastic discrete actor-critic. Authors in SDAC propose a general offline framework for RLRSs. They first formulate the recommendation problem as a probabilistic generative model. Then, a stochastic actor-critic algorithm is proposed to optimize the recommendation policy.

**Reward Formulation**

As depicted in Fig. 4.8(b), the majority (60%) of DRL-based RSs belong to R2. A common pattern frequently used by RLRSs in R2 is to formulate the reward as a function, or a simple combination, of several factors or metrics [151, 153, 5, 157, 159, 170, 169, 165, 178, 195, 185, 196, 187, 192, 191, 193, 210, 208, 204]. For example, in a news recommendation scenario, reward in DRN [5] is a function of user click and user activeness. The rationale behind factoring in the user activeness is that a good recommendation should motivate the user to use or interact with the system again. Authors use *survival models* [221] to model user return [222] and user activeness. FeedRec [157] formulates the reward as a weighted sum of instant metrics, including user click and purchase, and delayed metrics, like browsing depth and dwell time. Authors consider user clicks as instant metric, and browsing depth and return time as delayed metrics. Yu et al. [159] design an advantage function composed of visual, attribute, and history matching rewards, in order to tackle the multi-modal recommendation
problem.

Robust DQN [89] proposes to use *approximate regretted reward* to improve reward estimation. The idea is to use two different rewards, i.e., current and optimal rewards, and then calculate the regret as the final reward. Since calculating the optimal reward in the real is not possible, they propose to use an alternative, benchmark reward, which is the average reward achieved by applying the model to a subset of users.

A simple but effective scheme for multi-objective optimization in RLRSs is to formulate the reward as a multi-objective function [171, 181, 194, 197]. Singh et al. [194], for example, use this idea for a safe RLRS. The format of reward in their work is as follows

\[ R_{mo} = R_t - C_{\text{risk}}, \]  

where \( C_{\text{risk}} \) is a health risk constraint. This reward function balances between reward maximization and health constraint preservation. A similar formulation is used to balance the accuracy trade-off with diversity [171, 197] and fairness [181].

In hierarchical RL, two reward functions, i.e., for low-level and high-level agents, should be defined [152, 161, 201, 180, 206, 200]. For example, in HRL-Rec [206], click times on the recommended channel is considered as the low-level agent’s reward, while the reward for the high-level agent is composed of four factors, including click times, dwell time, list-level diversity, and item novelty.

In KGRE-Rec [8], a delayed reward function is used. Authors discuss that it is impossible to define a sparse, binary reward when there is no pre-known good/targeted item in their recommendation problem. Instead, the agent is encouraged to find good paths in the graph, those that lead to an item of user interest with high probability. Thus, the agent is received a reward only in a terminal state. The same idea can be seen in MASTER [203], where a *lazy reward* is given to the agent when a charging request is successful. However, the idea of rewarding the agent only in the terminal state is not always practical. For example, Liu et
al. discuss that since there is no well-defined terminal state in AnchorKG [204], the reward function should be composed of immediate and terminal rewards.

Another reward formulation method used in R2 is to define the reward as a distance between the recommended item and a target item [177, 186, 91]. In KGRL [186], for instance, the reward is based on the distance between the predicted item and target item in the graph

\[ r = \frac{100}{\sqrt{d(v_p, v_t) + \epsilon}} \cdot W_{pt}, \]  

where \( d(v_p, v_t) \) is the distance between predicted item \( p \) and target item \( t \), \( \epsilon \) is a regularizer, and \( W_{pt} \) is the sum of weights of the shortest path from \( v_p \) to \( v_t \). Distance \( d \) is calculated using Dijkstra’s algorithm.

On the other hand, perhaps the simplest method of reward definition for the designer is to empirically use several real values for different goals in the system, which is usually used in R1. For example, Zhao et al use a similar pattern of numerical reward in their proposals [2, 92, 86, 184] and reward three behaviors of users, namely skip, click, and order, with some numbers, e.g., 0, 1, and 5, respectively. The same pattern can be observed in other RLRSs belonging to R1 [152, 154, 158, 156, 164, 179, 175].

Similarly, in a conversational RS scenario, EAR [182] defines a sparse reward function with four pre-defined values, i.e., strongly positive reward when the recommendation is successful \( (r_s) \), a positive reward if the user gives positive feedback on asked attribute \( (r_a) \), a strong negative reward for user quit \( (r_q) \), and a slight negative for every conversation turn \( (r_p) \). The total reward is the sum of these rewards and in the experiments they use values \( r_s = 1, r_a = 0.1, r_q = -0.3, \) and \( r_p = -0.1 \). A similar reward function is used in other conversational RSs [160, 188, 205].

Delayed reward is also used in R1 by some graph-based RLRSs [173, 190, 202], where agent is only rewarded when it reaches the terminal state. For example in Ekar [173], the agent is rewarded with +1 if it reaches an item in the terminal state with which the user has
interacted, 0 if it reaches an item but the user has not interacted with, and -1 if the entity reached is not an item in the graph.

**Environment Building**

As shown in Fig. 4.8(c), more than half of DRL-based RSs use an offline method for environment building. Almost 40% of methods use a simulator, and only in 10% online study is used. Compared to RL-based methods, while a similar proportion use the offline method, the uptake of simulation and online schemes has been doubled and diminished by almost 60%, respectively. This graph shows that conducting an online study has become more difficult or costly, and simulation is getting more and more popular among the RLRS community.

Among those conducting a simulation study, SlateQ [120] introduces an open-source RLRS simulation environment, called RecSim [26], which gives the researcher flexibility to evaluate their algorithms in different settings. Cascading DQN [172] uses GANs to simulate a real user and estimate the reward function from logged data. More precisely, the GAN training is formulated as

$$\min_\theta \max_\alpha \left( \sum_{t=1}^{T} r_\theta(s_{\text{true}}^t, a_t^t) - \frac{R(\phi_\alpha)}{\eta} \sum_{t=1}^{T} r_\theta(s_{\text{true}}^t, a_{\text{true}}^t) \right),$$

where $\eta$ is a regularization term, $\text{true}$ means real data, $\phi$ represents the generator and generates user’s next action, and $r$ is the discriminator trying to differentiate between generated actions and real actions.

In DEERS [2], a user simulator, with the same architecture as DEERS, is trained on user logs. However, the output layer of the simulator is a softmax layer to predict the user feedback (immediate reward) based on the input (pair of state and recommended item). Authors claim that the simulator is 90% precise in predicting user feedback. The same approach for simulation study has been used by other RLRSs [92, 184, 4, 195, 196, 157, 207, 183, 180, 159, 166, 174]. For instance, a similar idea is used in [157], but the simulator (S Network) provides different feedbacks, including user response, dwell time, revisited time,
and a binary indicator if the user is leaving or not. In Pseudo Dyna-Q [174], a world model (user simulator) is trained by minimizing an error between online and offline rewards. _Truncated importance sampling_ [223] is used to alleviate the bias in the offline data.

Another popular simulation method is to develop a simulator based on collaborative filtering [86, 163, 198]. To be specific, LIRD [86] builds a memory with \((s, a, r)\) tuples seen in the logs dataset and uses a similarity method, based on cosine similarity, to find the closest state-action pair to the current state and action recommended. DRR [163] and DRGR [198] use the same intuition but based on probabilistic matrix factorization [141] and matrix factorization, respectively.

Building a simulator for conversational RSs is more challenging than that for typical recommendation scenarios mentioned above, as there are a small number of public datasets available to have both user rating and natural language/user chat for that item-rating pair. CRM [155] tackles this problem by creating simulated users based on Yelp [224] data and a dialogue corpus, collected using _crowd sourcing workers_. The simulated users have three behaviors: answering the agent question, finding the target item in a list, and leaving the dialogue. The same scheme has been used in other conversational RLRSs [205, 182, 188, 160].

Generally speaking, performing a good online study has become more challenging in modern RSs with huge user and item spaces, as the risk of implementing a non-optimal RS is very high. As discussed before, this is the most probable reason of a considerable decrease in the popularity of online study among DRL-based methods compared to RL-based methods. Perhaps two of the best online studies among RLRSs are conducted in [6] and [120] and performed on YouTube.

### 4.4 Emerging Topics

Having reviewed RLRSs, we have recognized that there are a couple of trends that are being formed among DRL-based RSs and have the potential to become mature in the course of
Multi-agent RL. Multi-Agent RL (MARL) is a generalization of a single-agent RL and is formulated as a Markov/stochastic game [225, 226]. MARL enables RLRSs to target several or complex tasks by dividing them into sub-tasks and each agent can handle one of them. From a game theory prospective, MARL methods can be generally divided into three groups: fully cooperative, fully competitive, and a mix of the two [226].

Recently, several RLRSs have employed MARL to tackle the problems of scholarly collaborator recommendation [134], mention recommendation in Twitter [165], page-wise recommendation [183], whole-chain recommendation [184], and charging spot recommendation [203]. As stated earlier, actor-critic with centralized training and decentralized execution has been a popular framework for DRL-based RLRSs employing MARL [165, 184, 183, 203]. In a cooperative setting [165, 183, 184, 203], a challenge is to determine the role of each player in the overall’s team success. CROMA [165], with two actors and a centralized critic, tackles this problem by a differentiated advantage scheme using reverse operation. Specifically, each actor agent can estimate its particular advantage by subtracting the overall Q value of the joint action, computed by the centralized critic, from the Q value of a reverse action. A similar architecture is used in DeepChain [184] to jointly optimize the overall reward of a session. It is not, however, clear how DeepChain solves the aforementioned problem, i.e., shared reward for two actors, which is critical for their effective training. In MASSA [183], a MARL with separate actor and critic agents is used to tackle a multi-module, page-wise recommendation. A game theory concept called correlated equilibrium [227] in the format of a signal network is used to handle the communication between agents. MASTER [203] considers each charging spot for electric vehicles as a distributed agent and uses a centralized critic to coordinate these agents. A couple of techniques, including bidding game and multiple critics, are employed to address challenges like cooperation between agents, future competition between requests, and multi-objective optimization. In a different, competitive scenario, authors in [134] use MARL to recommend scientific collaborator. Each author
looking for a collaborator is deemed as an agent and learn an optimal policy using gradient value iteration algorithm.

**Hierarchical and meta-controller RL.** Hierarchical RL (HRL) was initially sought to address the scalability problem in traditional RL algorithms [228]. In HRL, however, it is possible to define multiple layers of policies, each of which can be trained to provide higher levels of temporal and behavioral abstractions, leading to the ability of solving more complex tasks [229, 230]. Recommendation is not an exception and several researchers have utilized HRL in the RS domain [152, 161, 201, 180, 206, 200]. Generally speaking, all these RLRSs define a HRL with two levels of hierarchies where a high-level agent defines a high-level/abstract goal and a low-level agent tries to satisfy that goal. CEI [152] builds a conversational RS on a deep HRL method [231], which uses ideas from a popular and traditional HRL framework, called *options* [232]. CEI uses a meta-controller that selects a goal (chitchat or recommendation) in a given state and a controller makes an action following a goal-specific policy to satisfy the defined goal. Zhang et al. [161] employ HRL for course recommendation in massive open online courses (MOOCs). The key idea is to develop a profile reviser using HRL, which removes noisy courses from users profiles. This is decomposed into two high-level and low-level tasks: given a user profile and a target course, should the profile be revised (high-level) and if yes, which courses in the profile should be removed (low-level). DARL [201] improves Zhang et al.’s RS by making the recommendation unit more adaptive. That means, they equip the basic recommendation module in Zhang’s work with an attention mechanism to take dynamic users’ interest in diverse courses into account. HRL-Rec [206] uses HRL in an integrated recommendation scenario. A low-level agent generates a list of channels, and a high-level agent recommends a list of items with the channel constraint selected by low-level agent. Moreover, MaHRL [180] tackles the sparse conversion metric in e-commerce by using HRL. More precisely, there is a high-level agent responsible to track long-term sparse conversion interest by setting multiple abstract goals for the low-level agent, while the low-level agent follows these goals and tries to catch short-term
click interest. Finally, DHCRS [200] tries to tackle the large action space in RSs through using a two-level HRL, where a high-level DQN selects categories of items and a low-level DQN selects an item in the category to recommend.

In an emerging topic, a group of researchers have used RL as a meta-controller module in conversational RSs. That means, instead of using RL to optimize the recommendation policy, similar to HRL, these methods use RL to select either recommending items or asking questions from users to refine recommendations. But different from HRL, there is only one level using RL and the recommendation unit uses other techniques, like supervised learning, to generate the recommendations. This is the common theme in a couple of RLRSs [155, 188, 169, 160, 182, 187, 205]. For instance, CRM [155] is composed of three main parts: a belief tracker, a recommender, and a policy network (RL module). The belief tracker unit is responsible to extract facet-value pairs (some constraints) from user utterances and convert them to beliefs using an LSTM network. Factorization machine [233] is used in the recommender to generate a set of recommendations. Finally, a neural policy network, optimized by REINFORCE, is used to manage the conversational system, i.e., to decide either to ask for more information from the user or to recommend the items.

**Knowledge graph based RLRSs.** Incorporating knowledge graphs into RSs can boost recommendation accuracy and explainability [84]. Utilizing knowledge graphs provide RLRSs with different useful information, which can address sample inefficiency in DRL. Recently, many researchers started to use this idea and boost recommendation performance and explainability [8, 173, 193, 192, 191, 186, 204, 202, 185, 205, 190]. For example, the idea in KGRE-Rec [8] is to not only recommend a set of items, but also the paths in the knowledge graph to show the reason why the method has made these recommendations. An example of this graph reasoning is depicted in Fig. 4.16. For a given user A, the algorithm should find items B and F with their reasoning paths in the graph, like \{User A → Item A → Brand A → Item B\} and \{User A → Feature B → Item F\}. Obviously, graph based techniques face the scalability problem as the number of nodes and links can significantly
grow, proportional to the number of users and items. To address this problem, KGRE-Rec proposes a *user-conditional action pruning* strategy, which uses a scoring function to only keep important edges conditioned on the starting user.

**Supervised RL.** The key feature that distinguishes RL from supervised learning is whether the training data serves as an evaluation signal, like numerical reward, or as an error signal [234]. However, these methods, RL and supervised learning, can be combined to improve policy learning when both signals are available. Wang et al. [154] use this idea to dynamically recommend treatment options to patients. The idea is while the model should maximize the expected return, it should minimize the difference from doctors’ prescriptions. In particular, in an actor-critic architecture, the actor is responsible to recommend the best prescription by optimizing the following objective function:

\[
J(\theta) = (1 - \alpha)J_{RL}(\theta) + \alpha(-J_{SL}(\theta)),
\]  

where \(J_{RL}(\theta)\) and \(J_{SL}(\theta)\) are the objective function of RL and supervised learning tasks, respectively, and \(\alpha\) is a weight factor. Similarly, Liu et al. [195, 196] leverage supervised learning to guide the RL module in learning better policies. More precisely, in [195], a supervised learning signal helps generate better embeddings for state representation, and in [196], a supervised learning model is trained to guide the RL policy to focus on short-term reward and to generate top-aware recommendations.
Imitation Learning and Auxiliary Tasks. In addition to the above emerging topics, there are some topics, although less popular compared to the ones discussed above, we think that they have the potential to become emerging topics in the future. These topics include adversarial RL/training, safe RL, self-supervised learning, and imitation learning.

Adversarial training using GANs is an interesting emerging topic used in [168, 187, 190]. As mentioned earlier in Actor-Critic Methods section, CRSAL [187] and ADAC [190] use adversarial training integrated with actor-critic architecture for better agent training. Moreover, as discussed in policy gradient methods, IRecGAN [168] proposes a model-based RL based on GANs for the purpose of variance reduction and sample efficiency.

In safe RL, it is important for the agent to respect some safety constraints, along with maximizing the long term reward [235]. In Ref. [194], an RS based on multi-objective safe RL is proposed to improve the long term well-being of users. In particular, the agent simultaneously tries to maximize user engagement and the health of worst-case user.

Self-supervised learning (SSL) empowers the model to utilize labels available freely with the data. In [189], a framework is introduced to augment RLRSs with SSL. More precisely, authors propose a framework with two heads: RL and SSL. While the RL head is used as a regularizer to tune recommendations, the SSL head provides negative samples to update parameters.

In imitation learning, the agent is trained to perform a task from demonstrations [236]. Zhang et al. [162] combine imagination (model-based RL) and imitation learning to recommend personalized search stories. They argue that the goal of imitation learning is to imitate the policy of a recommender agent from which the logging data has been collected. Also, some fictional sessions are imagined by the agent and saved in a separate memory, and are used to fine-tune the agent training.
4.5 Open Research Directions

Slate Recommendation. RL algorithms have been originally developed to select a single action, e.g., the action with the highest Q value, in each time step from different actions around [237]. However, in the RS field, similar to many sequential decision support systems [237], it is wise to recommend a slate or list of items and let the user involved in the decision making process choose the best action, as the final goal is typically user satisfaction and recommendation acceptance. Despite some efforts [150, 120, 6, 172, 86], current RL algorithms cannot handle this problem. There are only two studies [150, 120] in the RLRS field that deeply investigate this problem. Slate-MDP [150] tries to solve this problem by searching the policy space for each slot in the slate individually. SlateQ [120] proposes to calculate the combination of the action set and consider each combination as an action. Slate-MDP cannot guarantee any optimality, and SlateQ is only applicable in two-stage RSs and fails to scale to single stage RSs with large action spaces. More attention is necessary in this aspect and more studies with solid theoretical foundations should be conducted in the future.

Explainability. Explainable recommendation is the ability of an RS to not only provide a recommendation, but also to address why a specific recommendation has been made [84]. Explanation about recommendations made could improve user experience, boost their trust in the system, and help them make better decisions [238, 239, 240]. Explainable methods could be generally divided into two groups: model-intrinsic or model-agnostic [241]. In the former, explanation is part of the recommendation process, while in the latter, the explanation is provided after the recommendation is made. An intrinsic explanation method could be the method we reviewed earlier [8]. On the other hand, as a model-agnostic example [242], RL is used to provide explanation for different recommendation methods. In particular, the method uses couple agents; one is responsible to generate explanations and another one predicts if the explanation generated is good enough for the user. One interesting applica-
tion of explainable recommendation is in debugging the failed RS [242]. That is, through explanations provided, we can track the source of problems in our system and to see which parts are not working properly. Although there have been some efforts in RLRSs to provide explainable recommendations [8, 173, 190, 204, 202], there is still a lack in this aspect and more attention is required in the future.

**Design.** All RLRSs reviewed employ RL/DRL algorithms that have been originally developed in domains other than RSs, like games [110, 117, 102]. These methods are typically designed based on physics or Gaussian processes, not based on complex and dynamic nature of a human. While sticking to available, cutting-edge RL algorithms and adapting them for RLRSs is wise, sometimes thinking out of the box could make a substantial improvement in the field. For example, instead of usual MDP-based RL algorithms, Ref. [243] uses *evolution strategies* [244] to optimize the recommendation policy, or Ref. [245] borrows ideas from a different literature and adapts them to the recommendation problem. Relevant to this, as surveyed in [17], there are many deep learning models developed for RSs. Because deep learning and DRL are closely related, perhaps wisely combining these models with traditional RL algorithms could outperform existing DRL algorithms. Last but not least, as illustrated earlier, some RL algorithms like Q-learning have been more popular among RLRSs than other RL algorithms. Nonetheless, there is no clue or justification behind the use of a specific RL algorithm for an RS application. Therefore, this would be a great study to possibly find a relationship between the RL algorithm and the RS application.

**Environment and Evaluation.** Fig. 4.17(a) depicts the most popular metrics by RLRSs. As shown, there is no metric specifically developed for RLRSs and almost all metrics are borrowed from Information Retrieval field. Although RSs and Information Retrieval fields are very close, but they are eventually different fields. Reward is also among the popular metrics used by RLRSs, which is a common metric in the RL literature. This analysis shows that there is a lack of metrics specifically designed for RLRSs. On the other hand, Fig. 4.17(b) illustrates the most popular datasets used to evaluate RLRSs. MovieLens
Figure 4.17: Evaluation metrics and datasets used more often by RLRSs

is the most popular dataset by far. A large number of datasets used for evaluation are not defined or public (see Tables 4.2 and 4.3). This limits the application and design of RLRSs to only a few applications, like entertainment. Therefore, it is important to collect and share more datasets in various domains to better evaluate RLRSs. Another aspect in RLRSs evaluation that needs further improvement in the future is to have a stronger and more unified simulation environment, something that can play the role of a benchmark in RLRSs’ evaluation. As stated earlier, online evaluation is the natural method to evaluate RLRSs; however, it is difficult and costly to conduct a proper online study. On the other hand, offline environment, i.e., a dataset, is static and biased. Therefore, this clearly shows the importance of developing a strong, general-purpose simulator for RLRSs, something like OpenAI Gym [246] in the RL literature. Although some environment simulators have recently been developed for RLRSs [247, 248, 249, 250, 251], this trend should be continued and fortified.

Reproducibility. The effect of reproducibility on the advancement of a field is undeniable. For example, the field of image synthesis using GANs has seen astonishing results in a short period of time [252, 253, 254], and undoubtedly, an effective factor has been the common practice of sharing implementation codes, datasets, and research results. As illustrated in Tables 4.2 and 4.3, we cannot see this trend in the RLRS research community and only about 16% of researchers have shared their implementation codes. It would be helpful and
can significantly accelerate the field’s progress if researchers accurately present the value of important parameters and hyperparameters used in their experiments, to perform statistical significance testing for results presented, to disclose which random seeds have been used to repeat experiments, and to share their implementation codes and datasets (if datasets are not already public).

4.6 Conclusion

In this paper, we presented a comprehensive survey on state-of-the-art in RLRSs. We highlighted the important role of DRL in changing the research direction in the RLRS field, and accordingly, classified the algorithms into two general groups, i.e., RL- and DRL-based methods. Then, we proposed a framework for RLRSs with four components, i.e., state representation, policy optimization, reward formulation, and environment building, and surveyed algorithms accordingly. Although many RLRSs have been proposed recently, we believe that the research on RLRSs is still in its infancy and needs plenty of advancements. Both RL and RSs are hot and ongoing research areas and are of specific interest to giant companies and businesses, so we can expect to witness new and exciting models to emerge each year. In the end, we hope this survey can assist researchers in understanding the key concepts and help advance the field in the future.
Sample Efficiency in Deep Reinforcement Learning based Recommender Systems with Imitation Learning

Abstract

Recently, there has been an unprecedented interest in using reinforcement learning (RL) for recommender systems (RSs), due to its unique ability in taking into account the dynamic and long-term user engagement. However, sample inefficiency is a major challenge in applying RL to problems with very dynamic environments and huge actions spaces. In this paper, we present Imitation, Reinforcement learning based Recommender System (IR2S) to combine RL with imitation learning to alleviate this problem. More specifically, by utilizing demonstrations (available user ratings), we show that IR2S can optimize its behavior faster and more efficiently. The proposed IR2S, built on top of Deep Q Network (DQN), shows superior performance compared to baselines in experiments.
5.1 Introduction

With the unprecedented increase in the volume of data on the web, information filtering techniques, like recommender systems (RSs), seem as a quite necessary part in our everyday lives. It is not uncommon to see that users experience the problem of *information overload* on a daily basis, which makes it tough for them to find their items of interest. By definition, RSs are software tools and filtering techniques that help users find their items of interest through predicting their ratings on items [61, 16]. Today, the application of RSs is widespread in a variety of domains, including movie, music, news, and e-commerce [255].

Traditionally, there have been two main methods widely used in the RS domain: collaborative filtering (CF) and content-based filtering (CBF). While the idea in CF is that similar users probably have similar interests [256], CBF’s idea is that users probably like items similar to those they positively rated in the past [99]. Recently, deep learning models, due to their ability in learning nonlinear interactions between users and items, have also been employed to recommend better items [17]. Nonetheless, one major problem with these methods, i.e., CF, CBF, and deep models, is that they are static and fail to take into account the dynamic and long-term user engagement [257].

Most recently, reinforcement learning (RL) has gained the attention of researchers in the RS field [257]. RL is a machine learning field that studies problems and their solutions in which agents, through interaction with their environment, learn what to do in order to maximize a numerical reward [25]. The ability of learning from a reward signal makes RL specifically a perfect match for RSs in which, in many cases, the user rating is scarce and the RS should learn to recommend better items using interactions with users. Moreover, despite previous methods—i.e., CF, CBF, and deep learning methods—RL is able to take into account dynamic and long-term user engagement.

Nonetheless, a major problem with applying deep RL (DRL) in the real is *sample inefficiency*. In other words, in light of complex environment dynamics and sparse reward
functions [258], it takes an unbearable time for the RL agent to receive non-zero rewards and to garner useful experiences. While probably acceptable in simulations, it is not practical in many real applications like RSs, where the user of the systems is a human and random exploration should be minimized [77].

Some RL-based RSs (RLRSs) have tried to tackle this problem by either model-based RL [172, 168, 174] or using supplementary data, like knowledge graph, for model-free models [8, 173, 193, 192, 191, 186, 204, 202, 185, 205, 190]. Model-based RL could be sample efficient as it uses its model for the purpose of exploration. However, estimating a good model especially in RSs with a very dynamic environment is not an easy task, if not unachievable. Another method is to use knowledge-graphs. Although knowledge graph-based methods have a high reliability, developing accurate knowledge graphs is difficult and cumbersome. Also, scalability easily becomes a problem in knowledge graph-based methods as the item and state spaces grow.

We tackle this problem by using imitation learning (IL) and demonstrations. More specifically, we propose Imitation, Reinforcement learning based Recommender System (IR2S) to combine RL with IL and show that this combination can be used to solve the exploration and sample inefficiency problem in RLRSs. We assume demonstrations are available in the form of user ratings, from a different recommendation setting, and adopt neighborhood CF methods to extract useful data from them to teach the RL agent. With experiments conducted in an offline study, we show that IR2S is effective and can considerably outperform baselines.

The remaining of this paper is organized as follows. Section 5.2 reviews the related work. We describe the proposed IR2S in section 5.3. Experiments are presented in section 5.4 and the paper is concluded in section 5.5.
5.2 Related Work

5.2.1 Collaborative Filtering

The key idea behind CF is that if two users are similar (they have rated items similarly in the past), they will probably rate new items similarly [256]. CF is specifically useful when there are ratings of users on items available. CF methods are usually divided into neighborhood approaches and latent factor models [256, 259]. The latter tries to uncover latent features that justify observed ratings. Matrix factorization [260] is the most popular technique from this family. On the other hand, neighborhood approaches, which are used in our proposed IR2S, focus on the relationship between items (item-based) [261, 262] or between users (user-based) [263, 264, 265]. In item-based methods, those items are recommended to a user which are similar to the items similarly rated by the user in the past. On the other hand, in user-based approaches, the interest of a user to an item is usually evaluated based on ratings given to that item by similar users. We adopt neighborhood methods to use in IR2S in light of their advantages compared to latent factor models, including simplicity, efficiency, and scalability [256].

5.2.2 RLRSs and Sample Inefficiency

While the first attempt to use RL for RSs, WebWatcher [122], is almost simultaneous with the foundation of RSs, it is just recently that many RLRSs have been proposed [179, 192, 177, 4, 209, 197, 200]. The reason of this recent interest is the foundation of DRL, a combination of deep learning and RL, which made it possible to apply RL in large action and state spaces [66, 65].

As stated earlier, sample inefficiency is a serious problem in applying DRL to many domains including RSs. There have been some attempts in RLRSs to alleviate this problem and they could be divided into two groups: model-based techniques and using supplementary
data for model-free models. An immediate solution is to use model-based RL [172, 168, 174], where the idea is to learn a model of the environment and reward function, and then use them to boost or accelerate agent’s learning. However, estimating a good/precise model especially in RSs with a very dynamic and complex environment (i.e., users) is not a trivial task, if not unachievable. In the latter, available data is usually used in the form of knowledge graphs [8, 173, 193, 192, 191, 186, 204, 202, 185, 205, 190] or auxiliary tasks [211]. A problem with knowledge graph-based methods is that the process of building a knowledge graph itself is cumbersome. More importantly, scalability is a potential problem in graph-based methods, specifically in RSs with a huge amount of users and items. In contrast to existing work, we propose to combine RL with IL. We use neighborhood CF methods to augment our RL agent. This considerably accelerates agent’s learning and, we believe, can solve the sample inefficiency problem.

5.2.3 Combining RL with IL

In IL, an agent tries to learn a mapping between observations and actions from demonstrations [236]. Behavior cloning and inverse RL are two forms of IL [258]. Our approach lies in the category of combining RL with IL. DQfD [266], DDPGfD [267], and Ref. [258] are related works in this topic. In DQfD, DQN is combined with IL. In particular, the learning of the agent is accelerated using demonstrations. DQfD introduces a margin loss that encourages demonstrations to have higher Q values compared to other actions. The main contribution in DDPGfD is to insert demonstrations into replay buffer of DDPG agent and using prioritized experience replay to play them. Ref. [258] considers an additional buffer for demonstrations, uses an auxiliary loss to train the actor, and apply behavior cloning loss to account for sub-optimality of demonstrations. In contrast to these methods, the proposed IR2S does not use any additional loss function and trains the RL agent with its original loss function. Instead, it provides a flexible framework to trade-off between random exploration and utilizing demonstrations.
5.3 Methodology

In this section, we describe the proposed algorithm. First, problem formulation is presented. Then, we quickly explain DQN, the RL algorithm on which we base IR2S, and neighborhood CF, the method we adopt to use demonstrations. Finally, IR2S is explained.

5.3.1 Problem Formulation

We study a recommendation problem in which an RS seeks to maximize the user engagement/satisfaction by sequentially recommending items to them. We consider the user and items as the environment and the agent receives a reward from the user in each time step $t$ upon recommending an item. This sequential decision making problem can be formulated as a finite Markov decision process (MDP) as follows.

- **State $S$:** a state $s_t \in S$ is defined as the user preferences and their past history with the system, i.e., last $N$ items positively rated by the user.
- **Action $A$:** an action $a_t \in A$ is to recommend an item to the user at time step $t$.
- **Reward $R$:** the RL agent receives reward $r(s_t, a_t) \in R$ based on the user feedback on the recommendation provided.
- **Transition probability $P$:** transition probability $p(s'|s, a) \in P$ is the probability of transition from $s = s_t$ to $s' = s_{t+1}$ if action $a$ is taken by the agent.
- **Discount factor $\gamma$:** discount factor $\gamma \in [0, 1]$ is the discount factor for future rewards.

Given $(S, A, R, P, \gamma)$, the goal of the RL agent is to find a policy $\pi$ that maximizes the expected, discounted cumulative reward. In other words,

$$\max_\pi \mathbb{E}\left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right],$$

where $T$ is the maximum time step in a finite MDP.
5.3.2 DQN

DQN is a combination of Q-learning [103] with a deep neural network, called Q network that plays the role of Q function. In traditional action-value RL methods, the optimal action-value function $Q(s, a)$ is achieved through an iterative rule, known as Bellman equation,

$$Q(s_t, a_t) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a')|s, a]. \quad (5.1)$$

In practice, this method is impractical due to its computational complexity. Instead, it is better to parametrize the action-value function as $Q(s, a) = Q(s, a; \theta)$. In DQN, Q network is responsible for action-value approximation. This Q network can be trained by minimizing a loss function as

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right], \quad (5.2)$$

where $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$ is the target for iteration $i$ and is the distribution over transition $s, a, r, s'$ collected from the environment. Differentiating $L(\theta)$ in Eq. (5.2) with respect to $\theta$ yields the following gradient

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]. \quad (5.3)$$

It is computationally beneficial to optimize the gradient in Eq. (5.3) using stochastic gradient descent [111].

5.3.3 Neighborhood CF

As mentioned earlier, we combine RL with IL using methods from neighborhood CF. Let’s assume demonstrations (user ratings) are stored in a buffer and can be displayed in the form of a matrix indicated by $\mathcal{M}$, where rows are users, columns are items, and each element $R_{ij}$
in $\mathcal{M}$ is a rating given by user $u_i$ to item $v_j$,

\[
\mathcal{M} = \begin{bmatrix}
R_{11} & R_{12} & R_{13} & \ldots & R_{1n} \\
R_{21} & R_{22} & R_{23} & \ldots & R_{2n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
R_{m1} & R_{m2} & R_{m3} & \ldots & R_{mn}
\end{bmatrix}.
\]

Given $\mathcal{M}$, the RL agent is provided with a list of tentative actions ($L_0$) to select from in the exploration phase. We select neighborhood CF methods over latent factor models, so as two methods could be used to provide $L_0$. The first method is to use item-based CF and to find a list ($L_0^i$) of the most similar items to one of the items ($v$) liked by the user ($u$) in the past. Note that $v$ is randomly selected from items liked by $u$ in the past. Another method to provide $L_0$ is to use user-based CF. First, a set of most similar users to $u$ is returned. Then, one of these users is randomly selected and the most similar items ($L_0^u$) to $v$ are found. We use cosine similarity to measure the similarity between users and items. For instance, the similarity between two users $u_i$ and $u_j$ is calculated by

\[
sim(u_i, u_j) = \cos(\overrightarrow{u_i}, \overrightarrow{u_j}) = \frac{\overrightarrow{u_i} \cdot \overrightarrow{u_j}}{||\overrightarrow{u_i}|| ||\overrightarrow{u_j}||} = \frac{\sum_{v \in I_{u_i u_j}} R_{u_i v} R_{u_j v}}{\sqrt{\sum_{v \in I_{u_i}} R_{u_i v}^2} \sqrt{\sum_{v \in I_{u_j}} R_{u_j v}^2}},
\]

where $I_{u_i u_j}$ is the set of items rated by both users $u_i$ and $u_j$.

### 5.3.4 IR2S

Our aim is to make DQN sample efficient with minimum modification. Accordingly, the difference between DQN and IR2S is that, instead of random exploration, IR2S explores through $L_0$. In particular, IR2S uses $\epsilon$-greedy for exploration, but with probability $\epsilon$, it selects actions from $L_0$, and with probability $1 - \epsilon$ it exploits actions. The remaining operations in IR2S are the same as that in DQN. The pseudo-code of IR2S is given in Algorithm 2.
Algorithm 2: IR2S Training Pseudo-code

1. Initialize replay buffer to capacity $\mathcal{N}$
2. Initialize Q network with random weights
3. for session = 1, $M$ do
   4. Initialize state $s_0 = \{u_i, v_1, v_2, ..., v_n\}$
   5. for $t = 1, T$ do
      6. With probability $\epsilon$ select randomly from $L_0$, otherwise select
         $a_t = \max_a Q(s_t, a; \theta)$
      7. Observe reward $r_t$ from $u_i$
      8. $s_{t+1} = \begin{cases} \{u_i, v_2, ..., v_n, a_t\} & \text{if } r_t = 1 \\ s_t & \text{otherwise} \end{cases}$
      9. Store transition $s_t, a_t, r_t, s_{t+1}$ in replay memory
     10. Randomly sample a minibatch of transition from replay memory
     11. Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal,} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$
     12. Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ according to Eq. (5.3)
   5. end
4. end

5.4 Experiments

In this section, we present experiments conducted to validate the performance of IR2S. First, experiments setup is explained. Then, the results of experiments are given.

5.4.1 Setup

In an offline study, we evaluate our method on two publicly available datasets: MovieLens 100K (ML100K) and 1M (ML1M) [148]. In each dataset, last 10 items rated by each user are used for test and the remaining is used for training. We convert the explicit ratings of users in these datasets, which are in the range of 1 to 5, to implicit feedback of 0 or 1. In particular, the rating is converted to 1 if it is over 3, and 0 otherwise. Table 6.1 shows the statistics of the two datasets.

We use two baselines to compare with the performance of IR2S. The first one is the original DQN algorithm [111]. In addition, we implement DEERS [2] as the second baseline.
The main idea in DEERS is to take into account not only the positive ratings of the user, but also the negative ones. It feeds the Q network with two separate inputs: one for positive states and another for negative states. Although DEERS uses prioritized experience replay [116], for fairness, original random sampling is used in our DEERS baseline.

It is noteworthy to mention that all recommendation methods are implemented in Python and experiments are conducted on Advanced Research Computing (ARC) cluster at the University of Calgary. For DQN and IR2S, the same network architecture is used: two fully connected (FC) layers with 64 and 32 neurons and ReLU activations, and an FC layer with linear activation as the output with sizes 1682 or 3952, depending on the number of items in datasets. For DEERS, we use two streams of FC layers for positive and negative states with sizes 64, 32, and 16 neurons, and then these two streams are concatenated by a single 32-neuron FC layer and reach an output layer similar to DQN and IR2S. States are the embeddings, with size 32, of user ID and items, noting that each state encodes the information of last five items positively rated by the user ($N = 5$). In DEERS, the same number of items is used for negative states. For stability, we use gradient clipping (with clip value 1) and changing the update rate for target network. We use Adam optimizer [268] with Mean Squared Error loss. Detailed hyper-parameters values are presented in Table 5.2.

Two metrics HR@$k$ and NDCG@$k$ [269] are used for performance evaluation. Hit Ratio (HR) checks if the test item is among the top items. Normalized Discounted Cumulative Gain (NDCG) measures the position of this hit in the recommendation list, such that hits at higher ranks in the list receive higher scores. In the next section, we report HR@20 and NDCG@20. We use the offline test method described in [2] to evaluate the three methods.

### Table 5.1: MovieLens Datasets Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th># users</th>
<th># items</th>
<th># ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML100K</td>
<td>943</td>
<td>1,682</td>
<td>100,000</td>
</tr>
<tr>
<td>ML1M</td>
<td>6,040</td>
<td>3,952</td>
<td>1,000,209</td>
</tr>
</tbody>
</table>
### Table 5.2: Hyper-parameters Values

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>Memory size</td>
<td>1M transitions</td>
</tr>
<tr>
<td>(N)</td>
<td>5</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>32</td>
</tr>
<tr>
<td>Min history to start learning</td>
<td>1000 transitions</td>
</tr>
<tr>
<td>Decayed (\epsilon)</td>
<td>(1 \rightarrow 0.1)</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.01</td>
</tr>
<tr>
<td>Loss</td>
<td>Mean-squared error</td>
</tr>
</tbody>
</table>

5.4.2 Results

In this section, the results of experiments conducted are presented. First, a parameter study is performed to examine the effect of the length of \(L_0\) on IR2S. Then, the performance of IR2S is compared against baseline methods.

**Parameter Study.** In this section, we study the effect of different values of \(L_0\) on the performance of IR2S. Figs. 5.1 and 5.2 show the results of item-based IR2S on ML100K and ML1M. We show the results for different numbers of similar items \((I)\). We investigate the effect of \(I\) on metrics when it changes from 1 (pure CF) to the largest number of items available (fully random exploration). As observable in both datasets, the best results are achieved when \(I = 10\). The results become worse with the increase in \(I\). This makes sense as with random exploration, it takes a long time to find useful experiences. This is consistent with what we expected according to the sample inefficiency problem in DRL.

Figs. 5.3 and 5.4 illustrate the parameter study for user-based IR2S performed on ML100K and ML1M. To be clear, we illustrate the results for different values of similar users \((U)\) and items \((I)\), instead of \(L_u^0\). Similar to item-based, user-based IR2S has the best performance when \(I = U = 10\) in ML100K. However, in ML1M, while \(U = 1000\) and \(I = 1\) shows the best results, it is generally seen that the performance changes arbitrarily with change in the values of \(U\) and \(I\).

**Comparison with baselines.** In this section, the result of comparison between IR2S
Figure 5.1: Parameter Study. The effect of different values of $I$ (similar items) on HR and NDCG (ML100K).

Figure 5.2: Parameter Study. The effect of different values of $I$ (similar items) on HR and NDCG (ML1M).
Figure 5.3: Parameter Study. The effect of different values of $U$ and $I$ (similar users and items) on HR and NDCG (ML100K). Vertical labels on x-axis indicate $I$ values.

Figure 5.4: Parameter Study. The effect of different values of $U$ and $I$ (similar users and items) on HR and NDCG (ML1M). Vertical labels on x-axis indicate $I$ values.
Figure 5.5: Comparing performance between algorithms (ML100K)

and baseline algorithms, i.e., DQN and DEERS, is presented. In addition to best results achieved in the previous section, we also report the averaged results for item- and user-based IR2S. That means, for example, HR for IR2S-AvgItem in Fig. 5.5 is the average of HR results depicted in Fig. 5.1. In general, according to Figs. 5.5 and 5.6, the three versions of IR2S significantly outperform the baselines. In particular, as depicted in Fig. 5.5 for ML100K, IR2S-Best can outperform baseline algorithms by 70% in terms of HR, and by 80% (DEERS) and 60% (DQN) in terms of NDCG. It is noteworthy to mention that user-based IR2S performs better than item-based IR2S on ML100K. On the other hand in ML1M, apart from IR2S-Best, it is observable from Fig. 5.6 that the performance diminishes in all methods when the dataset becomes larger. This decrease is more tangible in user-based IR2S and the baselines. From this figure, we can conclude that IR2S-Best can outperform the two baselines by 160% and 400% in terms of HR and NDCG, respectively. Another observations is that, ironically, item-based IR2S performs better than user-based IR2S on ML1M.
5.5 Conclusion and Future Work

In this paper, we proposed IR2S, a method to combine RL with IL. The main idea behind IR2S is to utilize demonstrations (user ratings) to alleviate the sample inefficiency in DRL. The results of experiments have confirmed the superiority of IR2S compared to baselines.

This paper is a pilot study to empirically test the performance of IR2S. In the future, we plan to extend IR2S with adding a margin loss to enable IR2S select between demonstrations and exploration.
Balancing Similarity-Diversity Trade-off in Recommender Systems: a Probabilistic Approach

Abstract

Recommender systems (RSs) are software tools and algorithms developed to alleviate the problem of information overload, which makes it difficult for a user to make right decisions. Two main paradigms toward the recommendation problem are collaborative filtering and content-based filtering, which try to recommend the best items using ratings and content available. These methods typically face infamous problems including cold-start, serendipity, scalability, and great computational expense. We argue that the uptake of deep learning and reinforcement learning methods is also questionable due to their computational complexities and lack of interpretability. In this paper, we approach the recommendation problem from a new prospective. We borrow ideas from cluster head selection algorithms in wireless sensor networks and adapt them to the recommendation problem. In particular, we propose Probabilistic Recommender System (PRS), a probabilistic scheme for item recommendation. We present three versions for PRS. The basic PRS can maximize the diversity and recommends
all items in a predefined duration. Furthermore, we factor in the importance of items in the recommendation process, which significantly improves the recommendation accuracy. We also introduce a method that considers a heterogeneity among items, in order to balance the similarity and diversity trade-off. Finally, we propose a new metric to measure diversity, which emphasizes the importance of diversity not only from an intra-list perspective, but also from a between-list point of view in a sequential/interactive recommendation process. With experiments in a simulation study performed on RecSim, we show that PRS is effective and can outperform baseline methods.

6.1 Introduction

The massive volume of information on the web leads to the problem of information overload, which makes it tough for users to find their items of interest. Many technologies have been developed to help users in this aspect, among which recommender systems (RSs) have been very promising [270, 16]. RSs are software tools and filtering techniques that help users find their items of interest through predicting their ratings on items. Today, the application of RSs is widespread, ranging from e-commerce [12] to news [13], from e-learning [14] to healthcare [15].

Traditionally, there are two main methods used in the RS domain: collaborative filtering (CF) and content-based filtering (CBF). The idea behind CF is to find similar users and to generate recommendations based on the assumption that similar users have similar tastes. In contrast, CBF recommends items that are similar to the rating history of the user. These methods typically face severe problems, including cold-start, diversity, scalability, great computational expense, and low quality recommendation [16, 271]. Recent attempts to alleviate part of these problems include deep learning (DL) and deep reinforcement learning (RL) methods. Nonetheless, the uptake of DL methods is questionable due to their inherent problems, including data hungriness (this problem is specifically acute regarding the data
sparsity fact in the RS field [256]), uninterpretability, and high computational complexities [17]. While RL is a better option compared to DL (due its ability in modelling user dynamics, working with implicit feedback [272], and taking into account sequential features and long term user engagement [257]), it also suffers from sample inefficiency, unstable and hard to achieve convergence, uninterpretability, and heavy computation, because the core of deep RL is a DL model employed in a semi-supervised learning field [111, 117].

Wireless sensor networks (WSNs) consist of, typically, a huge number of tiny sensor nodes, which are spread within or close to a phenomenon to monitor and collect data [273]. Although the possibility of adopting energy harvesting technologies has been investigated for these networks [274], power efficiency is still the most important challenge in these networks, due to the limited computational power and capacity of sensor nodes. Today, WSNs is a building block in the internet of things (IoT) technology and are applied in many applications [275, 276]. Clustering is an effective method in WSNs to make them more energy efficient [277], which divides the network architecture into two layers: cluster heads and cluster members. Typically, cluster members sense their surroundings and send their data to cluster heads, which are responsible to aggregate this data and transmit them to a remote station, known as base station. The heavy and important task of being a cluster head makes cluster head selection the main problem in clustered WSNs, as cluster heads should usually be rich in some features, like remaining energy, compared to other nodes [278, 279].

With carefully studying both problems, namely item selection for recommendation in RSs and cluster head selection in WSNs, we recognized that they are somehow comparable. In particular, if we assume that items in RSs are similar to sensor nodes in WSNs, items features, e.g., quality, are similar to nodes features, like remaining energy, and high quality items are similar to cluster head candidates, then we can assume that item selection for recommendation is analogous to cluster head selection in WSNs. This analogy motivated us to prepare this paper and to investigate the possibility of applying methods from clustering in WSNs to the recommendation domain. Therefore, we approach the recommendation problem
from a new perspective and aim at solving the problems of existing RS methods—i.e., cold-start, diversity, scalability, great computational expense, and uninterpretability—using the idea of load balancing in clustered WSNs [280]. The key idea behind load balancing is to rotate the heavy task of being cluster head among all sensor nodes periodically, so as the load is fairly distributed among all nodes and the network longevity is improved. We adapt this idea to the recommendation problem.

More precisely, with the idea of load balancing in mind, we propose three variants for Probabilistic Recommender System (PRS). Similar to [280], basic PRS (B-PRS) uses a probabilistic scheme for item recommendation, which can maximize the recommendation diversity. To improve recommendation accuracy, we extend B-PRS to factor in the importance of items in the recommendation strategy and call it priority PRS (P-PRS). To maximize user engagement, a good RS should provide a balanced level of accuracy and diversity in recommendations (known as similarity and diversity trade-off [281]), something that P-PRS fails to provide. Therefore, we present H-PRS, which considers a heterogeneity among items and can provide a flexible framework to balance the similarity and diversity trade-off. Not only the proposed methods are fast and explainable, they can solve the cold-start problem and scale well to the number of users and items. We study the performance of the proposed methods using a simulation study performed on RecSim [26], an RS simulator developed by Google, and the results of experiments are promising. In general, our contributions are as follows:

1. We approach the recommendation problem from a totally new perspective and apply methods from cluster head selection problem in clustered WSNs. To the best of our knowledge, no one has examined this idea before.

2. We take one step further and incorporate items importance into recommendation, which significantly improves the recommendation accuracy.

3. We also extend PRS to incorporate the heterogeneity of items into the recommendation
mechanism, which aims at balancing the similarity and diversity trade-off.

4. A novel diversity metric is introduced, which takes into account the importance of between-list diversity.

5. Using a simulation study performed on RecSim, we validate the performance of the proposed PRS.

This paper is organized as follows. Section 6.2 provides a quick background to introduce the topics discussed in the paper. Section 6.3 explains the proposed method in detail. Section 6.4 presents the results of experiments and the paper is finally concluded in section 6.5.

6.2 Background

In this section, we introduce the main concepts used in the paper. We first provide a background on RSs and its popular techniques. Then, we explain LEACH [280], a popular clustering algorithm in WSNs on which our work is based.

6.2.1 Recommender Systems

CF has been the foundation of first RSs developed [16]. The idea was simple: people tend to rely on recommendations made by their peers [16]. In fact, this was the rationale behind the first RS, Tapestry [97], and they termed it as collaborative filtering. Later, this term was broadened to recommender systems to reflect two facts [96]: 1) the method may not collaborate with users, 2) the method may suggest interesting items, not filter them. Another approach toward the recommendation problem is CBF, where the idea is to recommend items similar to the user profile, which is a structured representation of user interests [98, 99]. These methods, i.e., CF and CBF, fail to provide effective recommendations due to their problems stated above—cold-start, diversity, scalability, great computational expense, and low quality recommendation. Not long ago, unprecedented successes in the
machine vision field encouraged RS researchers to employ DL methods in the RS field [282, 283, 284, 285]. However, as mentioned earlier, DL models are data hungry, incomprehensible or unexplainable, and require massive computational resources [17].

Recently, many recommendation techniques have been proposed using deep RL methods [95, 92, 2, 89, 157]. Applying RL to RSs, with huge action and state spaces, was not possible until the advent of deep Q network (DQN) [111], which is the combination of traditional Q-learning algorithm [286] and convolutional neural networks. The use of RL in RSs has several advantages compared to the previous methods, including CF, CBF, and DL; first and foremost, RL is an effective method to model the sequential and dynamic nature of interaction between users and the RS. Second, RL can effectively work with implicit feedback [272]. Finally, unlike myopic methods (e.g., CF, CBF, and DL), it can maximize long term user engagement. Nevertheless, deep RL methods also suffer from problems similar to DL models, including slow convergence, unexplainability, and great computational expense. In section 6.4, we compare the performance of PRS with a DQN-based RS.

In contrast to the previous work, the proposed PRS is simple, fast, scalable, requires no rating from users, and can recommend a diverse range of items. To the best of our knowledge, we are the first to propose a recommendation approach using ideas from clustered WSNs.

6.2.2 LEACH

Low-energy adaptive clustering hierarchy (LEACH) [280, 287] is a popular clustering algorithm developed for WSNs. As stated earlier, the main idea in LEACH is to ensure that all nodes in the network play the role of cluster head once in a predefined time, which is known as load balancing in the WSN literature [277, 278]. In LEACH, the network operational time is assumed to be divided into rounds and each round has two phases: setup and steady-state. In the setup phase, clusters are formed and in the steady-state phase, the data is transmitted to the base station. LEACH uses a probabilistic approach for cluster head selection. More precisely, it is assumed that all nodes are synchronized and can start the network operations
at the same time. At the beginning of each round, all the nodes compute a threshold as:

\[
T(i) = \begin{cases} 
\frac{p}{1-p \cdot \left( r \mod \frac{1}{p} \right)} & \text{if } i \in G \\
0 & \text{otherwise,}
\end{cases}
\]  

(6.1)

where \( G \) is the set of nodes that have not been cluster head in the last \( \frac{1}{p} \) rounds, \( p \) is the optimal probability of cluster head selection, and \( r \) is the current round. Then, they generate a random number between 0 and 1. If the random number is less than \( T(i) \), node \( i \) selects itself as a cluster head in round \( r \) and advertises its decision within its surrounding to form its cluster. Otherwise, the node remains as a regular node and joins the nearest cluster head. This way, it is guaranteed that all the nodes are selected as cluster head once in \( \frac{1}{p} \) rounds. After this duration, they are eligible to be selected as the cluster head again.

6.3 Methodology

6.3.1 Problem Definition

We study the recommendation problem in which an RS, in each time step \( t \), recommends a list of items \( L = (v_1, v_2, ..., v_k) \) to a user \( u \in U \), where \( v \in \mathcal{I}, k \in \mathbb{Z} \), \( N = |U| \), and \( M = |\mathcal{I}| \). Each user \( u \) interacts with the RS in a recommendation session, which composes of several time steps \((t_1, t_2, ..., t_\tau)\), where finite \( \tau \in \mathbb{Z} \). Note that, in each time step \( t \), the user selects either one item or none from \( L \). The length of each session depends on the time budget \( (B_t) \) of the user. We assume that when \( B_t \) is over, the user leaves the system and never comes back. Then, the RS can start a new recommendation session with a new user. We consider a simple, fixed reward signal, indicated by \( R \); on selecting an item from \( L \) by the user, the agent receives \( R = c \), where \( c \) is a constant number. Otherwise, i.e., when no item is selected from \( L \), \( R = 0 \). In fact, \( R \) reflects the similarity between user interest and recommended

\footnote{This assumption is based on the user behavior model discussed in [288].}
items. On the other hand, a good recommendation approach should provide a diverse range of items [281]. Our RS objective is to recommend a list of items to the user in each time step $t$, which not only maximizes $R$, but it can also provide a proper level of diversity.

### 6.3.2 The Proposed PRS

**Basic PRS (B-PRS)**

Similar to LEACH, B-PRS uses a probabilistic approach for item recommendation. However, unlike LEACH in which cluster head selection is distributed and each node individually comes to a decision on whether select itself to be a cluster head, the recommendation process in our system is a centralized scheme performed by the RS agent. Therefore, we can rewrite Eq. (6.1) as

$$T_b(i) = \begin{cases} p \cdot \left( \frac{t \mod \frac{1}{p}}{1-p} \right) & \text{if } i \in G \\ 0 & \text{otherwise,} \end{cases}$$

Eq. (6.2) guarantees that all the items are recommended to users once in $\frac{1}{p}$ time steps, but it fails to control the number of items to recommend in each time step (i.e., $k$). For example, with a small $M = 100$ and $p = 0.05$ (a popular value in WSN literature), all the items are recommended to the user in 20 time steps and average $k = 5$. However, we observed in experiments that $k$ could be very noisy, with as large as 20 or 30 in one time step and as small as one in another time step. This is not definitely acceptable in a sensitive field like RSs whose users are humans. Instead, we use a deterministic approach in PRS and the element that defines the number of items to recommend in each time step is a predefined $k$. 

---

109
While there is no optimal value for $k$, there are some studies that examine the effect of this number on user satisfaction and suggest that a $k \in [5, 10]$ could be a good choice [289, 57]. We simply let $p = \frac{k}{100}$ and the maximum number of items to recommend in one time step is $k$. We empirically discuss this in section 6.4.

**Priority PRS (P-PRS)**

A problem with LEACH in the WSN literature is that it does not factor in the importance of nodes in cluster head selection and selects them randomly, while some nodes may have higher remaining energy than others and it is better to select them as cluster heads. Similarly, in our recommendation problem, some items may be of higher quality or are more popular than others and it is wise to recommend them to the user more often. Accordingly, we incorporate items importance into B-PRS and call it P-PRS. The most straightforward way to do this is to directly factor in the quality of items into Eq. (6.2) as

$$T_p(i) = \begin{cases} \left[ \frac{p}{1-p \cdot (t \mod \frac{1}{p})} \right] \cdot Q(i) & \text{if } i \text{ in } G \\ 0 & \text{otherwise,} \end{cases}$$

(6.3)

where $Q$ is the quality of item $i$ and is defined by the designer. As it is possible that $Q(i) < 0$ (see section 6.4), a problem with Eq. (6.3) is that $T_p(i)$ could be negative. Moreover, for $Q(i) > 1$, it is possible that $T_p(i) > 1$. To fix these problems, we first normalize $Q(i)$ using

$$Q_{\text{norm}}(i) = \frac{Q(i) - Q_{\text{min}}}{Q_{\text{max}} - Q_{\text{min}}},$$

(6.4)

where $Q_{\text{min}}$ and $Q_{\text{max}}$ are the minimum and maximum quality of items, respectively, and are defined by the designer (more details in section 6.4). Then, Eq. (6.3) can be rewritten as

$$T_p(i) = \begin{cases} \left[ \frac{p}{1-p \cdot (t \mod \frac{1}{p})} \right] \cdot Q_{\text{norm}}(i) & \text{if } i \text{ in } G \\ 0 & \text{otherwise.} \end{cases}$$

(6.5)
Eq. (6.5) guarantees that items with a higher quality are more likely to be selected for recommendation.

**Heterogeneous PRS (H-PRS)**

One problem with P-PRS is that it might perform greedily and only high quality items get selected for recommendation. This negatively influences the diversity, and consequently, the satisfaction of the user. To solve this problem, we introduce H-PRS. Inspired by [290], H-PRS assumes that items in $\mathcal{I}$ are heterogeneous and divides them into low-quality items $\mathcal{I}_l \subset \mathcal{I}$ and high-quality items $\mathcal{I}_h \subset \mathcal{I}$, where $\mathcal{I}_l \cup \mathcal{I}_h = \mathcal{I}$ and $\forall i \in \mathcal{I}_l$, $\forall j \in \mathcal{I}_h$, $Q(i) < Q(j)$. Accordingly, there are two thresholds, $T_i$ and $T_h$, for item selection at $t$, where $T_i$ factors in the probability of low-quality items ($p_l$) into item recommendation and $T_h$ incorporates the probability of high-quality items ($p_h$). These probabilities are calculated as

$$p_l = \frac{p}{1 + \lambda \cdot f},$$

and

$$p_h = \frac{p}{1 + \lambda \cdot f} \cdot (1 + \lambda),$$

where $\lambda$ is the coefficient of heterogeneity and $f$ is the fraction of $\mathcal{I}_h$. For example, if $f = 0.1$ and $\lambda = 2$, $|\mathcal{I}_h| = \frac{M}{10}$ and $\forall i \in \mathcal{I}_h$ has two times more priority than $\forall j \in \mathcal{I}_l$. Consequently, there are two thresholds for item selection:

$$T_i(j) = \begin{cases} \frac{p_l}{1 - p_l \cdot \frac{1}{p_l}} & \text{if } j \text{ in } \mathcal{G}_l \\ 0 & \text{otherwise} \end{cases},$$

and

$$T_h(i) = \begin{cases} \frac{p_h}{1 - p_h \cdot \frac{1}{p_h}} & \text{if } i \text{ in } \mathcal{G}_h \\ 0 & \text{otherwise} \end{cases}.$$
Algorithm 3: Item Recommendation in PRS

1. initialization \((k = 5, p = \frac{k}{100}, \mathcal{G}/\mathcal{G}_l/\mathcal{G}_h \leftarrow \mathcal{I}/\mathcal{I}_l/\mathcal{I}_h, L \leftarrow \emptyset)\)

2. for all \(u \in \mathcal{U}\) do

   while \(B_t(u) \geq \ell_d\) do

   while \(|L| \leq k\) do

   for all \(i \in \mathcal{I}\) do

   if \(i \in \mathcal{G}/\mathcal{G}_l/\mathcal{G}_h\) then

   calculate \(T_b(i)/T_p(i)/T_l(i)/T_h(i)\) using Eqs. (6.2)/(6.5)/(6.8)/(6.9)

   if \(\text{rand}(0, 1) < T_b(i)/T_p(i)/T_l(i)/T_h(i)\) then

   add \(i\) to \(L\) and remove from \(\mathcal{G}/\mathcal{G}_l/\mathcal{G}_h\)

   end

   end

   end

   end

   end

   update \(B_t(u)\)

   calculate \(p_l/p_h\) using Eqs. (6.6)/(6.7)

   if \((t \mod \frac{1}{p_l}/\frac{1}{p_h}) = 0\) then

   \(\mathcal{G}/\mathcal{G}_l/\mathcal{G}_h \leftarrow \mathcal{I}/\mathcal{I}_l/\mathcal{I}_h\)

   end

   \(L \leftarrow \emptyset\)

end

H-PRS provides a flexible framework to select the level of diversity in recommendations. There are two parameters \(\lambda\) and \(f\) that affect the performance of H-PRS. The selection of \(\lambda\) depends on the application and the level of diversity required. Moreover, we control \(f\) by defining a quality threshold, \(Q_{th}\). In other words, \(i \in \mathcal{I}_h \iff Q(i) \geq Q_{th}\). In the next section, we empirically show the effect of \(\lambda\) and \(Q_{th}\) parameters on the performance of H-PRS. Algorithm 3 demonstrates the pseudo code of item recommendation in PRS—for the sake of brevity, it demonstrates the three versions of PRS.

6.4 Experiments

In this section, we present the experiments conducted to show the performance of PRS. We use RecSim [26] to examine the performance of the proposed method. It is noteworthy to
mention that all recommendation methods are implemented in Python and experiments are conducted on Advanced Research Computing (ARC) cluster at the University of Calgary.

6.4.1 Setup

RecSim is a configurable simulation platform for RSs developed by Google. It provides RS researchers with the opportunity to develop various models of users and RS dynamics. It is noteworthy to mention that while PRS is a generic RS, we examine its performance in a document recommendation use case, similar to [120]. Except for the differences mentioned below, we configure RecSim exactly similar to the simulation study conducted in [120] (described in section 6 of that paper). The differences between our setup and theirs are as follows. First, we assume that there are $M$ documents in the local database of our RS to recommend to the user. This is in contrast with the setting described in [120], in which they assume that at each $t$, $m$ candidate documents are drawn from a large corpus and then a list of size $k$ from these $m$ candidates are recommended to the user. Second, unlike [120], user interest is not observable to the RS in our setting, which is the case in the real. Third, we assume that with probability $0.5$ all users select no item from the list recommended across time steps (explained shortly). Other settings of the experiments and assumptions are exactly the same as that in [120]. For completeness, we briefly describe these settings below.

For statistical significance, there are $N = 5000$ users interacting with the RS and $M = 10,000$ documents in $\mathcal{I}$ for recommendation. Documents are categorized into $T = 20$ topics and each document $d_i \in \mathcal{I}$ is associated with only one topic. While it is possible to associate each document to multiple topics [157], we will not study this case in this paper. For simplicity, it is assumed that the length of all documents are similar and constant. Each document has an inherent quality ($Q$), which shows the attractiveness of the document to the average user. The quality is randomly distributed across documents. The topics are divided into high quality and low quality topics. Of $T$ topics, $\lfloor \frac{T}{3} \rfloor$ of topics are of high
quality and $\forall d_i \in I_h, Q(d_i) \in [0, Q_{\text{max}}]$, where $Q_{\text{max}} = 3$. The remaining topics are of low quality and $\forall d_j \in I_l, Q(d_j) \in [-Q_{\text{max}}, 0]$. Users interest ($I_T$) to topics is modelled as a real number $\in [-1, 1]$, where -1 is totally uninterested and 1 is fully interested. Thus, $I_T \in \mathbb{R}^T$ is a vector with length $T$ and each element shows user interest to a specific topic ($I_t$). It is possible that $I_t$ changes after consuming a document from a specific topic. This change is computed as $\delta = (-y|I_t| + y) \cdot -I_t$, where $y \in [0, 1]$ ($y = 0.3$ adopted in experiments). A positive change in user interest, i.e., $I_t = I_t + \delta$, is occurred with probability $\frac{(I_t+1)}{2}$ and a negative change, i.e., $I_t = I_t - \delta$, with $\frac{(1-I_t)}{2}$. The document utility for a user is modelled as $S(u, d) = (1 - \gamma)I_t(d) + \gamma Q(d)$. Each document consumed by the user has the potential to replenish the user budget $B_t$. This is called bonus and is calculated as $b = (0.9/3.4) \cdot \ell \cdot S(u, d)$, where $\ell$ indicates how many time steps it takes to consume a document ($\ell = 4$ in experiments). It is possible for a user to not select a document from $L$. This choice is called null item and is the $(k+1)th$ item in $L$. For simplicity, we consider a fixed probability $P(\bot | L) = 0.5$ for all users to select the null item across time steps. Selecting a null item also consumes $\ell = 1$ unit from $B_t$. User behavior in selecting a document from $L$ is modelled using user choice model. We use the general conditional choice model defined in [120]. For a complete and more detailed description of these settings and assumptions, refer to [120].

As stated in section 6.3.1, the user provides a numerical reward $R$ upon receiving the recommendation. We assume a fixed $R = 4$ when the user selects a document from $L$. Otherwise, $R = 0$. Table 6.1 presents the value of parameters used.

In order to compare the performance of the proposed PRS with other works, we have selected the following baselines. Since we are using RecSim to evaluate the performance, we should use methods that only work with implicit feedback. All CF, CBF, and DL methods are not suitable to use as the baseline, since there is no rating and content available to apply these methods. Accordingly, we have implemented the following baselines:

- **DQN.** We have implemented a DQN [111] agent with the following specifications.
Each state $s \in S$ encodes information about the history of the user (last $m$ documents selected by the user). In addition to documents indices, documents topics and qualities are encoded in the states. No context from user but user ID is used in state encoding.

Each action $a \in A$ is to recommend a list ($L$) of $k$ documents to the user. We use a $k$ nearest neighbor method for our DQN agent to generate lists of size $k$ as actions$^2$. The same neural network architecture as that described in [111] is used for the Q network. Other DQN parameters are listed in Table 6.2.

- **ε-greedy.** We have also implemented an ε-greedy agent with $\epsilon = 0.1$, which is a popular method in multi-armed bandits [291]. ε-greedy explores with the probability of $\epsilon$ and exploits with $1 - \epsilon$.

- **Random.** This agent randomly selects lists of documents from $I$.

In the experiments conducted, we use the following performance metrics:

- **Average reward.** This metric, indicated by $R$, measures the similarity between users

$^2$In SlateQ [120], each list $L$ is an action and the action space $|A| = \binom{M}{k}$. While this setting is possible in a two-stage RS in which the first stage narrows down $M$ to $m$ candidate items, where $M >> m$, it is not simply scalable in our use case as with a not very large $M = 1000$, we have $|A| = \binom{1000}{5} \simeq 8 \times 10^{12}$. 

<table>
<thead>
<tr>
<th>Table 6.1: Parameters Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>$N$</td>
</tr>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$k$</td>
</tr>
<tr>
<td>$Q_{\text{max}}$</td>
</tr>
<tr>
<td>$T$</td>
</tr>
<tr>
<td>$y$</td>
</tr>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\beta$</td>
</tr>
<tr>
<td>$\gamma$</td>
</tr>
<tr>
<td>$B_t$</td>
</tr>
<tr>
<td>$\ell_d$</td>
</tr>
<tr>
<td>$\ell_\perp$</td>
</tr>
<tr>
<td>$P(\perp</td>
</tr>
<tr>
<td>$R$</td>
</tr>
</tbody>
</table>
Table 6.2: DQN Hyper-parameters Values

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>m</td>
<td>10</td>
</tr>
<tr>
<td>Memory size</td>
<td>1M transitions</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>32</td>
</tr>
<tr>
<td>Min history to start learning</td>
<td>1000 transitions</td>
</tr>
<tr>
<td>Decayed $\epsilon$</td>
<td>$1 \rightarrow 0.1$</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.1</td>
</tr>
<tr>
<td>Loss</td>
<td>Mean-squared error</td>
</tr>
</tbody>
</table>

interests and documents recommended and is calculated as:

$$\mathcal{R} = \frac{\sum_{u=1}^{N} \sum_{t=1}^{T_u} R_t(u)}{\sum_{u=1}^{N} \sum_{t=1}^{T_u} 1}. \tag{6.10}$$

- **Average diversity**: This metric, indicated by $\mathcal{D}$, measures the variety between recommended items. Two parameters are used for diversity metric. The first one measures *intra-list similarity (ILS)* and is calculated as [292]:

$$ILS(L) = \frac{\sum_{i \in L} \sum_{j \in L,i \neq j} Sim(i,j)}{\sum_{i \in L} \sum_{j \in L,i \neq j} 1}, \tag{6.11}$$

where $Sim(i,j)$ is the cosine similarity between items $i$ and $j$. To check the similarity between documents in our use case, we consider two features for each document: document topic and document quality. For document quality, two documents $d_i$ and $d_j$ are considered similar if $Q(d_i) > 0$ AND $Q(d_j) > 0$ or $Q(d_i) < 0$ AND $Q(d_j) < 0$. In other words, $d_i$ and $d_j$ are similar if $d_i,d_j \in \mathcal{I}_h$ or $d_i,d_j \in \mathcal{I}_t$. This measure (i.e., $ILS$) only checks the similarity within a list of items; however, in a sequential interaction between the user and the RS, it is important to track the similarity between consecutive lists recommended. Accordingly, we introduce a novel parameter, called
between-list similarity (BLS), and is calculated as

\[
BLS(L_n, L_{n+1}) = \frac{\sum_{i \in L_n} \sum_{j \in L_{n+1}} S(i, j)}{\sum_{i \in L_n} \sum_{j \in L_{n+1}} C},
\] (6.12)

where \(S(i, j) = Sim(i, j)\) and \(C = 1\) if \(i \neq j\). Otherwise, \(S(i, j) = k\) and \(C = k\). These values have been adopted to specifically penalize the RS method if an item is repeated across lists of recommendations. \(D\) is then calculated as

\[
D = \frac{\alpha \cdot ILS + \beta \cdot BLS}{2},
\] (6.13)

where \(\alpha\) and \(\beta\) are weights to define the importance of each diversity parameter, namely \(ILS\) and \(BLS\). For convenience, we let \(\alpha = \beta = 1\). Finally, similar to Eq. (6.10), the average diversity is computed as

\[
\bar{D} = \frac{\sum_{u=1}^{N} \sum_{t=1}^{\tau_u} D_t}{\sum_{u=1}^{N} \sum_{t=1}^{\tau_u} 1}.
\] (6.14)

While we report the results of diversity according to Eq. (6.14) in the next section, we indicate the average diversity by simple \(D\) for convenience.

### 6.4.2 Results

In this section, the results of the simulation study conducted on RecSim are presented. All results are statically significant and within a 95% confidence interval.

#### Parameter Study

We first study the effect of different parameters on the performance of the three methods B-PRS, P-PRS, and H-PRS. Figs. 6.1 and 6.2 depict performance metrics for B-PRS. In general, it is observable from figures that when \(k\) controls the number of documents recommended to the user, the effect of \(p\) on \(R\) and \(D\) is almost negligible. Moreover, when \(k\) increases, both
\( \mathcal{R} \) and \( \mathcal{D} \) are improved, although this improvement is more tangible in \( \mathcal{R} \). In particular, the improvement in \( \mathcal{R} \) is around 5% with \( k \) changing from 5 to 15, while this number is around 0.2% for \( \mathcal{D} \). The reason of this improvement for \( \mathcal{R} \) is perhaps in light of the larger \( k \), the more likely that the documents of interest to the user to be among recommended items. Also, when \( k \) is larger, the probability of having a more diverse list grows, as more items from different topics and qualities could be among the recommended items. It is worthy to note that, in Fig. 6.2, a smaller \( \mathcal{D} \) score means a better diversity.

On the contrary, Figs. 6.3 and 6.4 illustrate that P-PRS is sensitive to the value of \( p \), such that \( \mathcal{R} \) and \( \mathcal{D} \) experience a steady plummet with the increase in the value of \( p \). A reason for drop in \( \mathcal{R} \) is perhaps the fact that when \( p \) is larger, according to Eq. (6.5), \( T_p \) is generally larger so as documents with lower qualities also have a good chance to get selected for recommendation. Also, a larger \( p \) shrinks the \( \frac{1}{p} \) duration, again increasing the chance for all documents with any quality to be eligible for recommendation. This affects both user interest and budget, leading to a diminished \( \mathcal{R} \). Fig. 6.4 demonstrates that a larger \( p \) can improve diversity, probably with the same justification mentioned for \( \mathcal{R} \), i.e., a more diverse set of nodes have the chance to get selected for recommendation. Also, similar to B-PRS, a
large $k$ can improve both $\mathcal{R}$ and $\mathcal{D}$. Compared to B-PRS, while $\mathcal{R}$ is improved by almost 100% with P-PRS, this improvement is at the cost of sacrificing $\mathcal{D}$, which diminishes around 60%. This high similarity between items in a list may bore the user and negatively affect the overall user satisfaction. While a larger $p$ can help a little, P-PRS fails to provide a flexible framework to balance the $\mathcal{R}$ and $\mathcal{D}$ trade-off. We solve this problem using H-PRS.

In H-PRS, not only $p$ and $k$, but also $\lambda$ and $Q_{th}$ affect the performance. Fig. 6.5 shows $\mathcal{R}$ for various values of $\lambda$ and $Q_{th}$ when $k = 5$ and $p = \frac{k}{100}$. We have considered extremes and varied $\lambda$ from 0 to 100,000, and also $Q_{th}$ from -2 to 2. As it is seen, the increase in the value of both parameters can improve $\mathcal{R}$. This increase is steady and converges at some point for different values of $Q_{th}$. The least improvement is for $Q_{th} = -2$, for which the increase in $\lambda$ slightly improves $\mathcal{R}$ and quickly becomes stable at 185 when $\lambda = 20$. On the other hand, when $Q_{th} = 2$, $\mathcal{R}$ dramatically grows and becomes stable at 450 when $\lambda = 10,000$, which improves $\mathcal{R}$ by 170% when $Q_{th} = -2$. However, this is only part of the story; diversity in H-PRS is significantly decreased with the increase in the value of $\lambda$ and $Q_{th}$, depicted in Fig. 6.6. More specifically, although $\mathcal{D}$ is below 0.3 when $Q_{th} \leq -1$, it reaches more than 0.55 when $Q_{th} \geq 0$ and $\lambda \geq 500$. This clearly shows the trade-off between $\mathcal{R}$ and $\mathcal{D}$ discussed
Figure 6.3: The effect of $p$ on $\mathcal{R}$ in P-PRS

Figure 6.4: The effect of $p$ on $\mathcal{D}$ in P-PRS
earlier. From these results, one may ask which values are the best for $\lambda$ and $Q_{th}$? While the response to this question generally depends on the designer and the application at hand, H-PRS provides a flexible framework to meet desirable similarity and diversity constraints. To better demonstrate this ability of H-PRS, we compare its performance with two values of $\lambda$ to that of other methods in the next section. Moreover, we observed the same pattern as B-PRS for H-PRS for the effect of $p$ and $k$ on the performance metrics (i.e., while H-PRS is insensitive to $p$, $R$ and $D$ moderately improve with a larger $k$), so we decided to not report repetitive results.

Comparison with Baselines

In this section, the results of experiments for comparison between methods are presented. As stated earlier, we include two versions of H-PRS, indicated as HP in results, with $Q_{th} = 2$ and two values of $\lambda = 50$ and 10,000. Fig. 6.7 illustrates $R$ for all algorithms when $k$ grows from 5 to 15. Among the algorithms, H-PRS with $\lambda = 10,000$ significantly performs better than other methods. On average, it outperforms other methods between 40 to 170%. The reason of this supremacy is that H-PRS, with $\lambda = 10,000$, recommends high quality
documents—those with \( Q > 2 \)— more often. These documents not only replenish \( B_t \), but also they can change user interests toward high quality topics. Apart from H-PRS, P-PRS performs the second best, with the same superiority reason. DQN and \( \epsilon \)-greedy perform slightly better than B-PRS and Random methods, as they have the learning ability. Both B-PRS and Random methods achieve almost the same \( R \); it makes sense because both methods recommend documents randomly. The last observation is that, as expected, all algorithms perform moderately better with a larger \( k \).

To see the effect of items scale, Fig. 6.8 depicts \( R \) for various values of \( M \). The same pattern for algorithms’ performance is seen here. While H-PRS and P-PRS considerably outperform other methods, their performance is not affected with a larger \( M \). This obviously shows the robustness and scalability of these methods. Among the methods, DQN and \( \epsilon \)-greedy are negatively influenced the most by the increase in the value of \( M \). This is justified with the fact that as these methods are learning methods, when \( M \) is too large, they can not learn the value of all items, performing almost similar to a random method.

Finally, Fig. 6.9 illustrates diversity for all the methods. For the sake of completeness, we show \( D \), \( ILS \), and \( BLS \). Other than DQN and \( \epsilon \)-greedy, \( ILS \) and \( BLS \) are almost similar.
Figure 6.7: Comparison between algorithms and the effect of $k$ on their performance

Figure 6.8: Comparison between algorithms and the effect of $M$ on their performance
in all methods, leading to a similar $D$ according to Eq. (6.13). On the other hand, $BLS$ is significantly larger than $ILS$ for DQN and $\epsilon$-greedy, leading to a large $D$ that almost equals that of H-PRS, $\lambda = 10,000$ and P-PRS. The reason of a large $BLS$ for DQN and $\epsilon$-greedy is that these methods perform mostly greedily and select the same set of documents to recommend to the users.

### 6.4.3 Discussion and Future Plan

Several observations can be made from results presented in section 6.4.2. First, while H-PRS with $Q_{th} = 2$ and $\lambda = 50$ can perform as good as P-PRS in terms of $R$, it improves diversity by about 25%. This can be concluded that one can tweak H-PRS to effectively balance the similarity and diversity trade-off for a specific application, which can maximize the user satisfaction in the long term [281]. Second, random methods, including B-PRS and Random, can yield the best diversity ($D < 0.3$, see Fig. 6.9). Third, while a method can have a reasonable intra-list diversity ($ILS$), it can perform quite poorly in terms of between-list diversity ($BLS$). This observation emphasizes the importance of taking into account $BLS$ in computing diversity in a sequential human-computer interaction.
Both P-PRS and H-PRS methods are built under the assumption that the RS knows the exact quality of items. This assumption, albeit strong, could be generally true in the real. For instance, human editors of a news agency can select high-quality articles for the *featured* tab of their news platform [293]. This knowledge about the quality of items could be achieved by defining several quality attributes and scoring items based on these attributes. Or we might ask users to explicitly rank a list of quality attributes and then each item is matched against the user preferential ranking, receiving a quality score. Another method to elicit the quality of items is to gradually learn it through interaction with the user [120]. In general, it is more consistent with reality if the system can work with a more relaxed assumption about the quality of items, learned through one of the methods stated. We leave this for our future work.

Finally, it is agreed that performance evaluation of RSs is cumbersome [16]. This is mainly because, in order to evaluate the performance, the best way is to interact with a real user to have their feedback on recommendations made [56]. However, this process is very costly. A popular alternative is to use simulation as we used in this paper and is widely used by researchers in the RS field [95, 92, 2]. Following our discussion above, if we determine the quality of items using one of the methods explained, we speculate that PRS can be used and is effective in practice as well. This is also in our future work agenda; i.e., we plan to examine the performance of PRS in an online study using real users.

6.5 Conclusion

In this paper, inspired by LEACH protocol from WSNs literature, we proposed a new recommendation approach, called PRS, which is a simple, explainable RS and can solve many RSs’ problems, including cold-start and lack of diversity. We presented three variants for PRS; B-PRS does not discriminates between items and uses a probabilistic approach for recommendation. P-PRS, on the other hand, factors in the importance of items and recommends
items with higher qualities more often. Finally, H-PRS considers a heterogeneity among
items and provides a flexible structure to balance the similarity and diversity trade-off. The
results of experiments performed on RecSim validated the effectiveness of PRS.
Conclusion and Future Work

This chapter provides a summary for research findings presented in this dissertation and presents several directions for the future work.

7.1 Thesis Summary

Our first objective was to design an intelligent agent/bot to provide a next-generation patent decision aid. Accordingly, in chapter 2, I leveraged multi-layered and multi-agent systems technologies and proposed a design with learning abilities, which could evolve in the course of time and provide various medical services. This design, called ALAN, has three layers: User-Interface (UI), Analytical Decision Making (ADM), and Data. The UI layer is responsible to effectively interact with patients through three agents: human interaction, personal assistant, and survey. The ADM layer (core layer) plays the role of brain in the system and is responsible for important tasks like learning patient preferences, decision making, and analyzing patient data. Finally, the Data layer handles queries relevant to storing and retrieving data.

To develop the Learning agent in ALAN, in chapter 3, I developed an article RS with learning ability. In particular, the proposed article RS, called KERS, is a combination of MABs and knowledge-based systems. KERS is not provided with any information about
user’s interests so it should automatically learn them through interaction with users. Hence, in addition to cancer patients information provisioning, the primary objective in KERS is to maximize user satisfaction with minimum agent’s exploration. To achieve this, I added a memory or user profile unit to memorize the most recent user’s interests. Also, KERS utilizes a knowledge-base to quickly find user’s interest with minimum exploration. KERS uses an on-demand exploration method; i.e., it triggers the exploration phase whenever it detects a change in user’s interest. Since online evaluation of RSs is a very expensive and difficult task, I developed a flexible user simulator in order to evaluate the performance of KERS in an interactive offline study. The simulator provides two options to simulate a real user: the number of topics the user is interested about and the duration the user is interested about them. With different experiment settings, I showed that KERS can outperform baselines.

In chapter 4, I presented the first and most comprehensive survey on the state-of-the-art in RLRSs. This survey studies the use of RL by the RS community from the beginning, WebWatcher [122], until the most recent works based on DRL. This survey makes several important contributions to the field; first, it emphasizes the role of DRL in reviving the use of RL in modern RSs. Basically, it divides RLRSs into before and after DRL foundation. Second, it proposes a framework with four modules: state representation, policy optimization, reward formulation, and environment building. This framework can be used to formulate any RLRS and can unify the advancement of the field. Third, it is self-contained; that means, it provides a thorough review of key concepts in the field. For example, it provides the reader with a fairly complete knowledge on RL algorithms used by RLRSs, so they do not need to have a background in RL. Fourth, it highlights important trends and emerging topics in the RLRS field. In each component of the proposed RLRS framework, I tried to discover important trends. Also, there are several emerging topics in the RLRS field that have the potential to become mature in the future, including multi-agent RL, hierarchical RL, knowledge graph based methods, and supervised RL. Finally, I propose several open directions that should be more investigated in the future. These directions include slate
recommendation, explainability, RLRSs’ design, evaluation, and reproducibility. For example, in terms of slate recommendation, I discussed that RL algorithms have been originally developed to select one single, best action for the agent. However, many applications like RSs can provide a list/slate of items/actions and an end user can select the best action that suits them. This is not still properly addressed in the RLRS field and should be discussed more in the future.

In chapter 5, I addressed an important problem of DRL when applied to real world problems, i.e., sample inefficiency, in which it takes a long time for the agent to find useful states and to garner non-zero rewards specially in dynamic environments like RSs. To alleviate this problem, I proposed a method called IR2S, which combines RL with imitation learning. In other words, assuming demonstrations are available in the form of user ratings, IR2S provides a flexible framework to explore either demonstrations using collaborative filtering techniques or randomly. I used neighborhood based collaborative filtering techniques over latent factor methods in light of their advantages including simplicity, efficiency, and scalability. Using an offline study performed on two publicly available datasets, i.e., MovieLens 100K and 1M, I showed that IR2S is able to considerably outperform baseline algorithms.

Finally, in chapter 6, I proposed a probabilistic recommendation approach from a totally new aspect. Having studied a considerable body of RS literature, I found that there are still some problems with existing RS methods unsolved, e.g., having an RS that is not only simple, explainable, and fast, but it can solve the similarity-diversity trade-off. This motivated me to propose PRS, a new probabilistic recommendation approach based on ideas from Computer Networks field. In particular, PRS borrows ideas from cluster head selection in clustered sensor networks and use them to recommend an accurate and diverse set of items to users. To the best of my knowledge, no one else has explored this idea before. I proposed three versions for PRS: B-PRS, P-PRS, and H-PRS. In B-PRS, there is no difference between items and all the items have equal chance to get recommended to users in a pre-defined period of time. In P-PRS, some items with a higher priority or importance (like popularity) have a
higher chance to get recommended to users. A problem with P-PRS is that important items are recommended more frequently and the algorithm tends to perform greedily. To address this problem, H-PRS provides a flexible framework to balance the similarity-diversity trade-off by considering a heterogeneity between items. I also introduced a new diversity metric, which takes into account the diversity of items recommended not only from an intra-list prospect, but it emphasizes the importance of between-list diversity in RSs with sequential interaction with users. To evaluate the performance of the proposed methods, I used RecSim, an RS simulator developed by Google.

7.2 Future Work

There are several exciting research directions to expand in the future. As discussed earlier, I evaluated the performance of proposed KERS in an offline, simulation study on two relatively small-scale datasets. A possible future work direction is to test the performance of KERS in an online study using real patients and to screen the result of patient satisfaction in a long clinical study. Also, KERS does not use any context from users, items, and user-system interaction. It is interesting to see how a context-aware version of KERS performs compared with the context-free KERS.

The proposed IR2S in chapter 5 is an interesting combination of RL with imitation learning using collaborative filtering techniques. It is worthwhile to be investigated in the future a modified version of IR2S equipped with a marginal loss, that is able to automatically distinguish between demonstrations and explored actions and to take better actions with respect to the Q value of actions. Moreover, it is interesting to see how IR2S performs when implemented on a policy based RL method, like DDPG [117].

Finally, in PRS, discussed in chapter 6, it is assumed that the designer is aware of the importance or quality of items beforehand and this importance could be used to prioritize items in P-PRS and H-PRS. However, this might not be the case in all applications and it is
better for the algorithm to be able to figure out the importance of items through interaction with users in the course of time. Hence, it is a valuable research direction to equip the PRS with a mechanism to utilize the feedback/reward from the user and improve its behavior incrementally.
Bibliography


[5] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recom-


[170] Haokun Chen, Xinyi Dai, Han Cai, Weinan Zhang, Xuejian Wang, Ruiming Tang, Yuzhou Zhang, and Yong Yu. Large-scale interactive recommendation with tree-


