5-2022

Modeling and debiasing feedback loops in collaborative filtering recommender systems.

Sami Khenissi
University of Louisville

MODELING AND DEBIASING FEEDBACK LOOPS IN COLLABORATIVE FILTERING RECOMMENDER SYSTEMS

By

Sami Khenissi
M.Sc., Computer Engineering and Computer Science,
University of Louisville, Louisville, KY

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy in Computer Science and Engineering

Department of Computer Science and Engineering
University of Louisville
Louisville, Kentucky

May 2022

MODELING AND DEBIASING FEEDBACK LOOPS IN COLLABORATIVE FILTERING RECOMMENDER SYSTEMS

By

Sami Khenissi
M.Sc., Computer Engineering and Computer Science,
University of Louisville, Louisville, KY

A Dissertation Approved On

April 07 2022

by the following Dissertation Committee:

Dr. Olfa Nasraoui, Dissertation Director

Dr. Nihat Altiparmak

Dr. Hichem Frigui

Dr. Juw Won Park

Dr. Cara Cashon

ACKNOWLEDGEMENTS

I would like to thank my advisor and mentor Dr. Olfa Nasraoui for her guidance, mentorship and support. She believed in us when she took us in her lab five years ago and provided us with immense support and guidance. She taught us to always aim for excellence and never settle for anything less. We owe this milestone to her.

I want also to thank my committee members: Dr. Nihat Altiparmak, Dr. Hichem Frigui, Dr Juw Won Park and Dr. Cara Cushon for taking the time to review my work and provide very insightful feedback contributing to the quality of this dissertation

I want also to think the Computer Science and Engineering Department and University of Louisville and the head of Department Dr. Wei Zhang for their financial support and providing me the opportunity to pursue my degree and for their assistance throughout the last 5 years.

I would also like to acknowledge the National Science Foundation for partially supporting my research through grants IIS-1549981 and CNS-1828521.

I would like to thank my wife Mariem for her patience and support. She is a best friend, a great wife and a perfect colleague. Our long research conversations were the most important source of inspiration for all my research ideas.

I want also to thank my friends who shared this journey with me and offered unconditional help every step of the way.

I would like to thank my parents. Without their sacrifice, nothing would have been possible. I will be forever grateful. I want also to thank my little sisters Rim and Nour. They are the joy of our little family and their support means the world.

Finally a special thanks to my cousin and sister Sara who took me to my first day

of school 22 years ago, when I was a clueless little 6 years old child. She then set me on a path of excellence through her tutoring, support, and guidance during my early days.

To everyone that made this possible, I am forever grateful

ABSTRACT

MODELING AND DEBIASING FEEDBACK LOOPS IN COLLABORATIVE FILTERING RECOMMENDER SYSTEMS

Sami Khenissi

April 07 2022

Artificial Intelligence (AI)-driven recommender systems have been gaining increasing ubiquity and influence in our daily lives, especially during time spent online on the World Wide Web or smart devices. The influence of recommender systems on who and what we can find and discover, our choices and our behavior, has thus never been more concrete. AI can now predict and anticipate, with varying degrees of accuracy, the news article we will read, the music we will listen to, the movies we will watch, the transactions we will make, the restaurants we will eat in, the online courses we will be interested in, and the people we will connect with for various ends and purposes. For all these reasons, the automated predictions and recommendations made by AI can lead to influencing and changing human opinions, behavior and decision making. When the AI predictions are biased, the influences can have unfair consequences on society, ranging from social polarization to the amplification of misinformation and hate speech. For instance, bias in recommender systems can affect the decision making and shift consumer behavior in an unfair way due to a phenomenon known as the feedback loop. The feedback loop is an inherent component of recommender systems because the latter are dynamic systems that involve continuous interactions with the users, whereby data collected to train a recommender system model is usually affected by the outputs of a previously trained model. This feedback loop is expected to affect

the performance of the system. For instance, it can amplify initial bias in the data or model and can lead to other phenomena such as filter bubbles, polarization, and popularity bias. Up to now, it has been difficult to understand the dynamics of recommender system feedback loops, and equally challenging to evaluate the bias and filter bubbles emerging from recommender system models within such an iterative closed loop environment.

In this dissertation, we study the feedback loop in the context of Collaborative Filtering (CF) recommender systems. CF systems comprise the leading family of recommender systems that rely mainly on mining the patterns of interaction between the users and items to train models that aim to predict future user interactions. Our research contributions target three aspects of recommendation, namely modeling, debiasing and evaluating feedback loops. Our research advances the state of the art in Fairness in Artificial Intelligence on several fronts: (1) We propose and validate a new theoretical model, based on Martingale differences, to model the recommender system feedback loop, and allow a better understanding of the dynamics of filter bubbles and user discovery. (2) We propose a Transformer-based deep learning architecture and algorithm to learn diverse representations for users and items in order to increase the diversity in the recommendations. Our evaluation experiments on real world datasets demonstrate that our transformer model recommends 14% more diverse items and improves the novelty of the recommendation by more than 20%. (3) We propose a new simulation and experimentation framework that allows studying and tracking the evolution of bias metrics in a feedback loop setting, for a variety of recommendation modeling algorithms. Our preliminary findings, using the new simulation framework show that recommender systems are deeply affected by the feedback loop, and that without an adequate debiasing or exploration strategy, this feedback loop limits the discovery of the user and increases the disparity in exposure between items that can be recommended. To help the research and practice community in studying recommender system fairness, all the tools developed to model, debias, and evaluate recommender systems are made available to the public as open source software libraries [1]. (4) We propose a novel learnable dynamic

[1] https://github.com/samikhenissi/TheoretUserModeling

debiasing strategy that learns an optimal rescaling parameter for the predicted rating and achieves a better trade-off between accuracy and debiasing. We focus on solving the popularity bias of the items and test our method using our proposed simulation framework and show the effectiveness of using a learnable debiasing degree to produce better results.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT v
LIST OF TABLES xii
LIST OF FIGURES xiv
LIST OF ALGORITHMS xv

CHAPTER

1 INTRODUCTION 1
1.1 Problem Statement 3
1.1.1 Scope and Impact of the dissertation 4
1.2 Research Contributions 6
1.2.1 Document Organization 7

2 BACKGROUND AND LITERATURE REVIEW 9
2.1 Recommendation Systems 9
2.1.1 Content-based Filtering Methods 10
2.1.2 Collaborative filtering Methods 12
2.1.3 Attention mechanisms in recommender systems 16
2.2 Bias in recommender systems 16
2.2.1 Diversity Bias 16
2.2.2 Popularity Bias 17
2.2.3 Exposure Bias 18
2.2.4 Iterative or Closed Feedback Loop Bias 19
2.2.5 Why should we care about the bias in recommender systems? . 19
2.3 Traditional evaluation systems 22

2.4 Summary 24

3 TRANSFORMER-PROMOTER NETWORK: A SELF-ATTENTION TRANSFORMER ARCHITECTURE TO PROMOTE DIVERSE RECOMMENDATIONS 27

3.1 Transformer-Promoter Network: Methodology and Design 28

3.1.1 Overview and Problem Statement 28

3.1.2 Promoter Network 30

3.2 Evaluation of the Transformer-Promoter Network 34

3.2.1 Datasets 34

3.2.2 Baselines 35

3.2.3 Implementation Details 35

3.2.4 Evaluation 36

3.2.5 Experimental Results 38

3.2.6 Learning Diverse Embeddings 40

3.2.7 Ablation Study 41

3.3 Summary 43

4 THEORETICAL MODELING OF USER DISCOVERY IN A FEEDBACK LOOP ENVIRONMENT 48

4.1 Theoretical Modeling of the feedback loop in a Collaborative Filtering System 48

4.1.1 Notation 48

4.1.2 Assumptions 51

4.1.3 Asymptotic behavior of human discovery 52

4.1.4 Effect of personalization on the human discovery 58

4.2 Validation of the Proposed Theoretical Model for Recommender System Feedback Loops 60

4.2.1 Ranking Assumption Validation 60

4.2.2 Empirical Validation of the Asymptotic Behavior of the User Discovery . . . 62
4.2.3 Effect of a greedy exploration approach on human discovery . . 64
4.3 Summary . . . 65

5 SIMBA: A MODULAR SIMULATION FRAMEWORK FOR STUDYING BIAS IN RECOMMENDER SYSTEMS . . . 69
5.1 SimBa: Design and Methodology . . . 69
5.1.1 Problem statement . . . 69
5.1.2 Proposed design . . . 70
5.1.3 Example: Simulation module for classic Matrix Factorization . 75
5.2 Demonstrating the use of the SimBa simulation framework for feedback loop analysis and dynamic debiasing . . . 78
5.2.1 Research Questions . . . 79
5.2.2 Data . . . 80
5.2.3 Debiasing strategies . . . 80
5.2.4 Bias evaluation metrics . . . 81
5.2.5 Measuring Both Relevance and Bias using the Expected Popularity Complement (EPC) . . . 83
5.2.6 Implementation details . . . 83
5.2.7 The joint effect of the Debiasing Strategies on the recommender system Feedback loop (RQ5.1 - RQ5.5) . . . 83
5.2.8 The effect of breaking the feedback loop through random recommendations (RQ5.6) . . . 85
5.2.9 The effect of the initial dataset bias on the feedback loop (RQ5.7) 87
5.2.10 Limitations and future work . . . 89
5.3 Summary . . . 89

6 L-DYNAMICS: ADAPTIVE DYNAMIC DEBIASING STRATEGY FOR

ITERATIVE RECOMMENDER SYSTEMS 92
6.1 A New Learnable Dynamic Debiasing Strategy based on T-O Market Principles 92
6.1.1 A theoretical analysis inspired by Trial-Offer markets 93
6.1.2 L-DynamicS: A learnable dynamic debiasing strategy for iterative recommender systems 97
6.2 Dynamic Debiasing Evaluation 100
6.3 Summary 101
7 CONCLUSION 104
REFERENCES 107
CURRICULUM VITAE 117

LIST OF TABLES

TABLE Page

2.1 Comparison between the proposed SimBias simulator and other simulators in the literature. 25

3.1 Summary of the notation used for the Transformer-Promoter Network. . . . 27

3.2 Statistics of the used datasets 44

3.3 Hyperparameters used for training the Transformer-Promoter Network . . . 44

3.4 Performance of diversity, novelty, and popularity bias in the recommendations with three different datasets. Significant results are in bold ($p - value < 0.05$) and comparable results are underlined. We observe that the Transformer-Promoter Net significantly outperforms all the other models. . 45

3.5 Performance of predicting accurate recommendations with three different datasets. Significant results are in bold (p-value ¡0.05), comparable results are underlined. We observe that the Transformer-Promoter Net has comparable results to the state of the art while having better rating prediction accuracy in the case of Movielens-1m 46

3.6 Comparison between Scaled Dot Product Attention and Bias-Aware Attention in terms of accuracy 47

3.7 Comparison between Scaled Dot Product Attention and Bias-Aware Attention in terms of diversity and bias 47

4.1 Summary of the notation used for the theoretical feedback loop modeling . 49

4.2 Comparison between the mean of the predicted ratings for items from the seen groups (same genre) and items from unseen groups. 61

5.1 Effectiveness of our simulation framework at testing research hypothesis compared to other available simulation frameworks 91
5.2 Hyperparameters used for MF and IPSMF for both ML100K and Yahoo! R3 datasets 91
6.1 Analogies between a T-O market and a dynamic recommender system scenario 102
6.2 Comparison of the EPC level at iteration 50 and percent increase of EPC for different debiasing strategies 103

LIST OF FIGURES

FIGURE Page

1.1 User embedding space representation example 3

2.1 Two-Tower Architecture example 11

2.2 Neural Matrix Factorization Architecture 14

2.3 Summary of the effects of bias on recommender systems agents 20

3.1 Transformer-Promoter Network architecture 28

3.2 Contextual Neighborhood illustration 29

3.3 Projection of the top 10 items embedding 40

3.4 Ablation study for the Transformer-Promoter Network 42

4.1 Illustrative example for an iterative recommendation process 50

4.2 Predicted rating distribution of the seen item groups vs unseen item groups 61

4.3 Experimental Results with the Perfect Feedback Assumption 4.1.2 66

4.4 Results with the Relaxation of Perfect Feedback Assumption 4.1.2 67

4.5 Results using a greedy exploration strategy with Multi-Armed Bandits. 5(a) 68

5.1 Modular decomposition of the recommender system framework 70

5.2 Simulation steps as implemented in SimBa 77

5.3 The evolution of the bias evaluation metrics for the Movielens data 84

5.4 The evolution of the bias evaluation metrics for the Movielens data while using periodic random recommendations 86

5.5 The evolution of the bias metrics for the Yahoo! R3 dataset 87

6.1 The evolution of the bias metrics for the Yahoo! R3 dataset. We notice that MAB outperforms other metrics in terms of EPC 102

LIST OF ALGORITHMS

ALGORITHM Page

3.1 Prediction steps for Promoter Layer for items 33

5.1 Full Simulation Steps 78

6.1 Learning the optimal debiasing degree in L-DynamicS 100

CHAPTER 1

INTRODUCTION

Gartner predicts that through 2022, 85 percent of AI projects will deliver erroneous outcomes due to bias in data, algorithms or the teams responsible for managing them [1]. One particular area where AI has been used is in recommendation systems. The goal of a recommendation system is to personalize the user's experience by suggesting content predicted to match their preferences. With the exponential growth of information and services online, recommender systems have become an essential Artificial Intelligence component of most web and smart device applications, as they promise to enhance the user experience when facing information overload. Recommender systems follow three major paradigms depending on the desired goal. Collaborative Filtering (CF) [2] relies on the user-item interactions in order to learn the similarities between the different entities, and is considered the most prevalent state of the art, in part because it does not require explicit encoding of the item content or user demographics. Content-based Filtering, on the other hand, utilizes the users' and items' content features to build predictive models for recommendation [3]. Finally, hybrid models combine different paradigms [4]. For instance they may rely on both user and item content features and Collaborative Filtering interaction data in order to provide recommendations.

Because recommender systems are intended to be a necessary tool to help users wade through the vast space of online information and to avoid information overload, they are increasingly becoming an unavoidable component of the increasingly AI-influenced society daily. Their influence permeates human experiences every time that humans interact with AI algorithms, often unbeknownst to them, as they perform various tasks online. These tasks affect a large spectrum of our life, such as learning, discovery, education, entertain-

ment, job seeking, e-commerce, and networking with others. Yet, these AI systems have been recently shown to feed on biased data and to generate and amplify bias that can lead to unfairness [5]. The term *bias* in recommender systems has been used to denote a wide, but often related, variety of biases [6]. These include popularity bias, diversity bias, exposure bias, display bias, iterative bias, etc. The propensity to be biased is especially the case for user-based and item-based collaborative filtering [5]. User-based CF assumes that users who have shared similar behavior or preferences in the past, will tend to behave (or have preferences) *similarly* in the future [7]. User similarity is usually defined based on their past interactions with the current items in the recommender system. In the same way, item-based CF relies on the similarity between how different items are interacted with by users, to infer future recommendations. For this reason, we focus on Collaborative Filtering in this work. Popular CF methods include Matrix Factorization [8] based models, Neural Matrix Factorization [9], Autoencoders [10], and Graph Neural Networks [11]. These models need user-item interaction data in order to be trained. For instance, Matrix Factorization (MF) operates by reconstructing a user-item interaction or rating matrix starting from an incomplete matrix where missing interactions or ratings are to be predicted.

Because collaborative Filtering techniques rely on past interaction data in order to infer recommendations, the methods used to collect these interactions are crucial for ensuring that the results are accurate and unbiased. In fact, once deployed in production, a recommender systems is considered a dynamic system where recorded interactions are used for training the models for the next iteration. Hence, recommender systems operate within an ecosystem that can be affected by multiple biases that originate from this dynamic iterative behavior [12]. These biases can have multiple consequences. They can reduce user satisfaction by recommending redundant and non-diverse items to the user [13]. They can also reduce fairness and worsen the inequalities between items by boosting certain items and suppressing other underexposed items [14]. Other effects include polarization and filter bubbles [12].

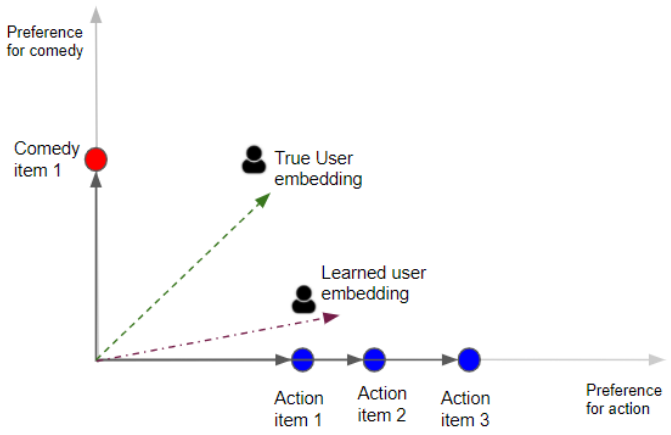


Figure 1.1. The figure illustrates a two-dimensional user embedding space where the first dimension represents a preference for comedy and the second one represents a preference for action. When the recorded interactions of the user are dominated by one type of item, the learned embedding will be heavily influenced by this type, which in this case, is action movies. We thus see that the learned embedding deviates from the true embedding of the user that would have been learned if the data was unbiased. The example refers to the movie domain, but the phenomenon is identical in other domains, ranging from books to news and music.

1.1 Problem Statement

One particularly successful and popular family of Collaborative filtering (CF) models comprises latent factor models which can be considered to rely on learning an embedding of the users and items using the user's past interactions, and then mapping these embeddings into predictions [15]. Matrix Factorization [8], for instance, learns the missing ratings by modeling the users and items as embedding vectors in a shared space and then uses the inner product as a scoring function. This will result in placing similar items and similar users close to each other in the embedding space. The problem with this kind of framework is the assumption that the recorded interactions fully represent the user's profile and tastes. Such full interaction data could be achieved by designing an unbiased data collection strategy such as randomly displaying to the user some recommended items and collecting ratings for them for several iterations. However, in the real world, such a strategy has generally been unfeasible or inefficient given the vast space of all possible user-item interactions and the relevance-focused goal of a recommender system, which by definition, is not supposed to recommend random items. In fact, the recorded ratings come

from previous user interactions with the recommender system. This *feedback loop* between the recommender system's outputs in one cycle and its inputs for the next cycle, creates a biased training dataset, where the user interaction might lack enough coverage to represent the full user profile, and this, in turn, misleads collaborative filtering models into providing less diverse recommendations, thus further biasing the output in a vicious cycle between biased input and biased output.

To further understand the effect of feedback loops on CF systems, let us consider the example in Figure 1.1. Suppose that user u has an equal preference to action and comedy movies. However, in the previous iterations of the recommender system, user u has rated three action movies and only one comedy movie. The recorded data does not reflect the true preference of this user as it has been affected by the redundancy of action movies as illustrated in Figure 1.1. Using this biased version of interaction data in the training process will bias the learned embedding of the user and the items toward favoring redundant and similar items. Hence the user might end up getting similar recommendations in every iteration. Furthermore, the iterative nature of the recommender system will exacerbate this bias with every iteration [16, 17].

1.1.1 Scope and Impact of the dissertation

We discuss feedback loops and bias in recommender systems from three main perspectives, modeling feedback loops, debiasing recommendation systems, and auditing recommendation systems for bias.

1.1.1.1 Debiasing recommender systems

We propose various methods for reducing the effect of feedback loop bias on the final recommendations. The idea is to develop new debiasing strategies that account for the impact of popular and over-exposed items. This will help increase the diversity of the recommendation list.

Debiasing recommendation models is crucial for reducing the effect of filter bubbles,

polarization and echo chambers which are affected by the feedback loop [12]. There is also potential benefit to reducing the bias of recommender systems because the latter are being used to filter content in many critical domains such as online news and jobs search [18, 19].

We start by providing a static debiasing methodology through a novel deep learning architecture. We then propose a dynamic debiasing strategy that learns an optimal debiasing degree to achieve a healthy balance between accuracy and popularity.

1.1.1.2 Feedback Loop modeling

We model the evolution of the feedback loop in order to understand its dynamics. We propose a theoretical model that describes the evolution of the recommender system and study the theoretical limits of the user discovery in a collaborative filtering recommender system. We then empirically validate our theoretical findings using real world data.

A theoretical model of the feedback loop is important because it will pave the way for developing better debiasing strategies that dynamically incorporate the feedback loop effect.

1.1.1.3 Auditing recommender systems bias impact in a Feedback loop environment

An other important area that we tackle is evaluating and auditing recommender systems while taking into account the challenging feedback loop setting in which they operate. This is crucial to go beyond the currently dominating and yet limited paradigm of offline evaluation of accuracy, which ignores the feedback loop; and to allow evaluating the bias impact of any recommendation model in a more realistic setting. The main challenge of this task is providing an accurate simulation of how recommender systems behave. The idea is to model the feedback loop while also controlling various factors that affect the feedback loop and investigating the evolution of bias in each iteration.

The ability to audit recommender systems in a feedback loop environment allows designing and studying new debiasing strategies by testing them in a dynamic environment.

Our simulation framework thus allows researchers to test their debiasing strategies in a more realistic and controlled environment without the need to go through expensive and risky online testing.

1.2 Research Contributions

This dissertation studies, through a theoretical and experimental framework, the critical problem of existing and emerging bias due to the phenomena of feedback loops in recommender systems. Bias from recommendation feedback loops poses a real threat to society since it may lead to filter bubbles [12], unfair commercial and labor opportunities [14], and to increased social polarization. Therefore, this dissertation research can have an impact on the research area of AI and recommender systems and has the potential to improve the fairness of recommender systems deployed in industry. This in turn can revolutionize the fairness of state of the art AI systems, permeating numerous areas of society from learning and entertainment to business and health management. More specifically, our research advances the state of the art in Fairness in Artificial Intelligence, through the following main contributions:

1. We propose a Transformers-based deep learning architecture that uses a novel regularized self-attention unit to provide diverse recommendations to the user even when trained on biased data.

2. We empirically show the effectiveness of our debiasing strategy on several real world recommender system benchmarking datasets.

3. We present a theoretical study of the evolution of feedback loops in recommender systems and we study the convergence of the user discovery in such a setting.

4. We empirically evaluate the claims proven in our theoretical study and we study the effect of our assumptions on the results.

5. We propose a simulation framework (SimBA) based on a modular decomposition of the

recommender system ecosystem that helps in studying the effect of each component in a dynamic feedback loop setting.

6. We use our simulation framework to study the effect of debiasing strategies on the recommender system feedback loop through several controlled experiments.

7. To help the research and practice community in studying recommender system fairness, all the tools developed to model, debias, and evaluate recommender systems are made available to the public as open source software libraries [1].

8. We propose a **L**earnable **Dynamic** Debiasing **S**trategy For Iterative Recommender Systems (*L-DynamicS*) based on Trial-Offer Market Principles, that takes into account the feedback loop effect in order to mitigate the popularity bias of the system without sacrificing accurate results.

9. We perform an empirical evaluation for our dynamic debiasing strategy using our proposed simulation framework. We show that our strategy, L-DynamicS, performs significantly better than the competing methods in providing accurate **and** less biased recommendations.

1.2.1 Document Organization

The rest of this dissertation is organized as follows. In Chapter 2, we provide a literature review of the different methods used in recommender systems and discuss previous bias research in terms of studying the bias and debiasing strategies, while highlighting the feedback loop effect. Then in Chapter 3, we study static debisaing by proposing the Transformers-based Promoter Network to improve the diversity of the recommendation list. We propose a comprehensive empirical evaluation on different real world datasets to evaluate our novel architecture.

In Chapter 4, we describe the theoretical study to model the feedback loop through studying the user discovery convergence. We provide empirical evaluation to our theoretical

[1] https://github.com/samikhenissi/TheoretUserModeling

results Then, in Chapter 5, we present a new modular simular framework to evaluate the dynamic bias in recommender systems. In Chapter 6 we propose an adaptive and learnable dynamic debiasing strategy in order to balance accuracy and debiasing in a feedback loop setting.

Finally, we make our conclusions in **Chapter 7.**

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

Recommender systems help with narrowing the choice space of users and direct their focus from millions or thousands of items to a handful of options to choose from. The main advantage of these systems is that they help maximize the utility of the user without spending too much time browsing a prohibitively large pool of items. Recommender systems are nowadays used prominently throughout our daily activities to influence our choice of what video to watch, to filter posts on social media, or influence how we browse news articles. Therefore there is a lot of interest in making these systems more accurate, effective, and less biased.

During the first part of this literature review, we will focus on enumerating techniques used in recommender systems. In the second part, we will focus on the bias of recommender systems and how the feedback loop can affect the performance and the evolution of the system.

2.1 Recommendation Systems

In this section, we describe the different models and paradigms used in recommender systems. Recommender system models can be categorized into three main groups [20]: Content-based Filtering models, collaborative filtering models, and hybrid models. Hybrid models are basically a combination of multiple methods, such as combining content-based and collaborative filtering, where the system uses item content features and user features along with collaborative filtering methods to create a more robust recommender system. We start by describing the different methods used in content-based filtering, then we enumerate models used in collaborative filtering. In the second part, we dive deeper into the problem

of bias in recommender systems and we discuss the different types of bias arising from the feedback loop and how the bias phenomenon has been studied in the literature.

2.1.1 Content-based Filtering Methods

Content-based filtering recommender systems rely on item features and user features to provide recommendations [21]. In fact, the problem can be treated as a classification problem where the model is trained to predict whether a user u is going to like or interact with item i based on their features. Another way to train such a system is to use ranking-based loss functions [22]. The latter learns an ordering relation between pairs or lists of items and the aim is to learn if user u prefers item i over item j.

To learn such patterns, the main idea is to represent the user and items in a feature space; then to use a classifier to predict a given similarity score or the probability of a user liking or interacting with a given item. This can be achieved using traditional feature engineering and then training a tree-based classifier on sparse features [23]; or we can transform the users and items into a common embedding space and then apply a neighborhood-based method such as K-Nearest Neighbors in order to predict the score of user-item pairs [24].

Using tree-based classifiers is straightforward. We can engineer different kinds of user features such as demographic features and item features such as the genre of the item, the popularity or recency of the item. Using these features, tree-based classifiers such as Gradient Boosted Decision Trees [25] (GBDT) can be used to find an ensemble of decision trees that classifies the user-item interactions [26, 27]. This technique has been proven to outperform many deep learning-based approaches on large-scale recommendation tasks [26]. A different way to leverage GBDT is to use ranking losses such as LambdaRank [28]. The loss helps optimize ranking metrics directly such as NDCG [29] by building trees to improve the ranking.

Another popular approach in content-based recommender systems is to project the user features and item features into an embedding space and use it to classify or rank interactions. In fact, many items have textual or visual features that can leverage recent deep

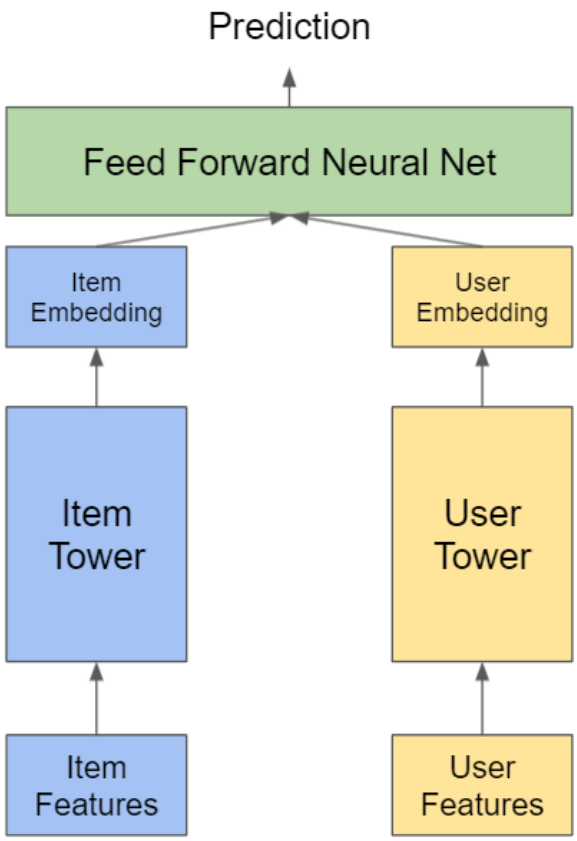


Figure 2.1. Example of a two-tower architecture or a Siamese Network. The diagram presents two neural network architectures where each network projects the sparse features into an embedding space. A feedforward MLP is then used to compute a score to the user-item pair interaction. It should also noted that we can replace the MLP layer with a simple dot product.

learning architectures such as Transformers [30] or Convolution Neural Nets (CNN) [31] in order to extract embeddings from these features. These features can then be concatenated to a user embedding that can be created by also leveraging similar deep learning architectures for textual features or a simple Multi-Layer Perceptron (MLP) to transform the sparse features into embeddings. The final embedding vector can then be fed into an MLP to classify the interaction [32].

A pipeline for the described approach is shown in Figure 2.1. In fact, embedding-based approaches can use Two Tower architectures [33] (also called Siamese Networks [34]). Where we use two deep learning architectures, one for users and one for items. Each tower

or network will project the users and items to an embedding space. Then finally we use these embeddings to classify or rank the interactions. The advantage of using such architecture is decoupling the user network from the item network. This helps in using the embeddings for several other tasks such as retrieval. Also, it helps accelerate the inference stage as these embeddings can be precomputed and stored in a key-value table which allows for fast retrieval during the inference stage [35].

The main advantage of content-based recommender systems is the ability to predict the likelihood of interaction for new items. On the other hand, the main disadvantage is the need to collect features for all users and items and this in turn can be expensive especially from the user side. The users usually do not enter the same amount of information which creates sparsity in the user feature map and this in turn can lead to inaccurate results.

2.1.2 Collaborative filtering Methods

Collaborative Filtering (CF) is another paradigm of recommender systems. It can further be categorized into item-based Collaborative Filtering and user-based Collaborative Filtering. It assumes that similar users (or items for item-based CF) will have similar future patterns. The main advantage of Collaborative Filtering is not needing any content features from the users or items and being able to provide accurate predictions using only user-item interactions. Many classes of models fall within this category, including Matrix Factorization-based models [8], Neural Matrix Factorization [9], Autoencoders [10], and Graph Neural Networks [11].

2.1.2.1 Matrix Factorization

Matrix Factorization (MF) is a Collaborative Filtering based model that takes as input an incomplete user-item interaction matrix and learns to make a reconstruction of such a matrix using a model consisting of latent representations of users and items.

More formally, let R be an interaction matrix where R_{ui} is a rating (or a given interaction) for user u to item i. Let $\mathbb{R}^{(\mathbb{K})}$ be a latent space of dimension K. Matrix

Factorization learns a user representation P and an item representation Q in the latent space the dot product $P.Q^T$. For an explicit feedback setting where the aim is to predict the rating of the user to the item, Matrix Factorization minimizes the objective function given by:

$$J(P,Q) = \frac{1}{|U||I|}\sum_{(u,i)}(p_u.q_i^T - R_{ui})^2 + \beta(||P_u||^2 + ||Q_i||^2) \tag{2.1}$$

Where β is a regularization hyperparameter to prevent overfitting. The minization of such cost function can be done using Stochastic Gradient Descent [36].

After learning the latent factors P_u and $Q-i$, we can then represent users and items in the same latent space $\mathbb{R}^{(\mathbb{K})}$. Therefore, items that are close to the user in such space are assumed to be relevant items. This similarity is captured through the dot product in the inference stage. items and users that have high dot products tend to be closer in the latent space.

Matrix Factorization (MF) presents many advantages. The first benefit of MF is the speed of training and inference. In fact, the input matrix is very sparse. This sparsity helps speed up the training process, when coupled with Stochastic Gradient Descent (SGD) [36]. The training can be parallelized [36] to obtain faster results. The inference stage relies on computing dot products. We can compute dot products faster by distributing the computation even for millions of items. Therefore, MF is typically used during the first stages of a recommender system to narrow the item space from millions of items to hundreds of items [37].

Another advantage of Matrix Factorization is that, being a collaborative filtering method, it only needs ratings. We do not need to collect other features from the users or items which can be an expensive operation.

A weakness of Matrix Factorization is the cold start problem. In fact, for a new user or a new item where no interactions are yet recorded, we cannot learn the missing ratings for this user or item. A solution for this is to query the new user for ratings by using Active Learning, for instance, in order to get seed signals to train the model.

Another weakness of this model is relying on the Missing At Random data assump-

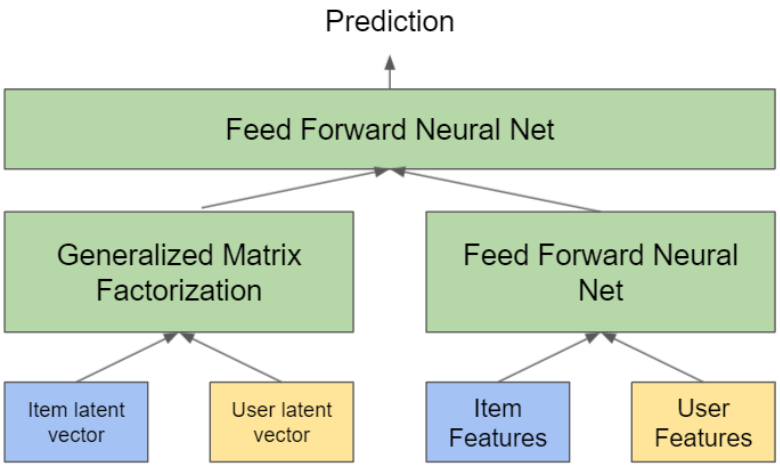


Figure 2.2. Architecture of Neural Matrix Factorization. We train four vectors: P and Q are users' and items' representations to be fed to a Generalized Matrix Factorization. U and V are users' and items' representation to be fed to an MLP architecture.

tion [38] so that it correctly estimates the missing ratings. However, the data is usually Missing Not at Random [39] due to different biases that are introduced. Techniques such as Inverse Propensity Weighting [38] can be used to mitigate the drawbacks of this issue.

There are several variants of Matrix Factorization, including Probabilistic Matrix Factorization (PMF) [40], Poisson Matrix Factorization [41], and Bayesian Personalized Ranking (BPR) [42] to cite a few. Each of these methods uses the same inductive bias by learning user and item representations within a shared latent space. They however learn these representations using different techniques. BPR, for example, uses a ranking-based loss function in order to include user preference while learning these representations. PMF uses a probabilistic approach to model prediction as the probability of user u liking (or interacting with) item i.

2.1.2.2 Neural Matrix Factorization

Another popular paradigm for learning user and item representation uses neural network architectures. For instance, Neural Matrix Factorization (NeuMF) [9] is a popular method that uses the same architecture as matrix factorization, but models the similarity of users and items using a Multi-layer Perceptron (MLP) layer instead of the dot product. [9]

argues that using MLP will enable learning non-linearity of the user and item interactions. The overall architecture is shown in Figure 2.2. NeuMF learns four different embeddings, two for users and two for items. First, the embeddings are learned similarly to Matrix Factorization but by replacing the dot product with an element-wise product. This is called the Generalized Matrix Factorization Layer. Another set of embeddings are learned through feeding user features and item features to an MLP layer. Finally, the results from both architectures are concatenated and fed to a final MLP architecture to predict the final score. All the embeddings in NeuMF are learned jointly from end to end.

A main advantage of NeuMF is that it can be extended to content-based systems by including user features and item features in the MLP layer. However, there has been an ongoing debate on the effectiveness of NeuMF compared to MF [15, 43]. In fact, recent research argues that careful tuning of MF-based architectures can outperform NeuMF [15]. It is also argued that the dot product is best suited to model user-item relationships, while feedforward architectures may fail to capture this interaction [15].

2.1.2.3 Autoencoder

The autoencoder was introduced as a Collaborative Filtering Model through AutoRec [44]. The idea is similar to other latent models which learn a different representation of users and items in a lower-dimensional latent space. Autoencoders achieve this compression by trying to reconstruct the input data (i.e ratings). The hidden neurons are then used as a latent space representation.

Other extensions to the Autoencoder include Variational Autoencoders (VAE) [45]. The aim of these methods is to learn a generative model that can generate the original input. VAE showed great promise in delivering competitive results but still suffers from several limitations such as Latent variable collapse [46]. This happens when the approximated posterior is independent of the data, resulting in poor latent variables that fail to represent the original data.

2.1.3 Attention mechanisms in recommender systems

Attention mechanisms have become omnipresent since the breakthrough results of the transformer network in NLP tasks [30], where it was shown that in order to learn meaningful dependencies, recurrent units were not needed and only attention layers are needed. Through the use of self-attention to attend to the same sequence, the architecture resulted in state of the art performance, only to be improved upon later and lead to new, more powerful transformer-based models, such as BERT [47], GPT-2 [48], and GPT-3 [49]. Following their success in NLP, efforts to apply self-attention mechanisms to other domains have increased. For instance, a recent work [50], adopting transformer layers to replace convolution layers, has produced promising results for computer vision tasks.

Following this trend, several efforts adopted the transformer architecture for recommender system tasks, with [51] using the BERT architecture for next item prediction, and [52] applying self-attention for sequential recommendations; while [53] extended the latter approach by including the item similarity to improve the user and item representations. Other work [54] used attention to map the user ratings to the reviews and proposed a new objective function to reduce popularity bias. Attention mechanisms were also applied in graph-based recommendation [55], retrieval tasks [56], and click through prediction [57].

2.2 Bias in recommender systems

The term *bias* in recommender systems has been used to represent a wide, but often related, variety of biases [6]. These include popularity bias, diversity bias, exposure bias, display bias, iterative bias, etc. We will therefore summarize the bias types that we consider to be the most related to our work

2.2.1 Diversity Bias

Diversity bias studies the variety between items within a recommendation list. This is probably the earliest type of bias studied, as it directly relates to the overfitting problem in machine learning [13]. Some of the prior work [58] focused on the diversity of recommen-

dations by incorporating different techniques to diversify the final recommendations. These techniques, including Determinantal Point Process [59], regularization techniques [60], optimization techniques [61], and so on, focus on the *model learning step* of the recommender system algorithm. In fact, by designing a method that effectively *learns* to promote the recommendation of diverse items, we can alleviate the problem of diversity bias.

Other methods have focused on post-processing methods by designing a new recommendation strategy based on exploration techniques, such as Multi-Armed-Bandits algorithms [62–66] or other post-filtering methods using personalized recommendation selections [67]. In fact, MAB adds exploration to the recommendation process and therefore increases the diversity of the recommendations. The main intuition of the MAB strategy is to sacrifice some short-term reward in order to increase the long-term outcome. Different MAB strategies have been applied to the RS setting. A popular method is the ϵ-greedy strategy because of the simplicity of its implementation.

Recently, researchers have adopted Reinforcement Learning (RL) techniques in order to learn an "intelligent" exploitation exploration trade-off [68, 69]. However, Reinforcement Learning techniques still face criticism due to their expensive evaluation process of the proposed policies in an online setting and the logging bias introduced in an offline setting [70]. In order to correct for the logging bias, Inverse Propensity Scoring has also been proposed in this setting [71].

Diversity bias causes several known issues in recommender systems and information retrieval such as filter bubbles and polarization [72, 73]. We argue that diversity bias itself can be a direct consequence from the iterative behavior or the closed feedback loop in recommender systems.

2.2.2 Popularity Bias

Another type of studied bias is popularity bias [14]. It often translates to studying the long tail items. In fact, popularity bias causes some unfairness between items, since popular items get increasingly promoted compared to other items [74].

In order to mitigate this bias, several techniques have emerged, ranging from regularization [75, 76] to re-ranking and post-processing techniques [77, 78]. These methods aim at promoting the long tail items in order to provide a fair exposure to these items and reduce unfairness.

Even though it is often distinct from iterated bias (Section 2.2.5), popularity bias is largely affected by the behavior of the closed feedback loop [17]. In fact, for a collaborative filtering algorithm where no external data is included in the training and recommendation process, popular items get recommended often because their high number of ratings helps the algorithm learn to group them more accurately. This bias phenomenon can be considered to be related to the cold start problem [75]. Popularity bias is exacerbated when the user's feedback is used in the next iteration of training the model.

2.2.3 Exposure Bias

We can think of exposure bias as the iterative bias (Section 2.2.5) studied in a *single* fixed iteration. In this setting, we study how the user was exposed to the items before providing the ratings, and then how this will affect the next predictions. This is equivalent to studying the Missing Not At Random (MNAR) problem [79]. In fact, by studying the distribution of the missing data, we can infer the effect of the bias on the predictions and/or the training.

Within this scope, previous studies have tried to estimate the probability that the user is exposed to an item before rating it, and then use this estimate to alter the training algorithm by designing better estimators of the performance of the algorithm [38, 39, 80, 81] or using regularization techniques [82].

Exposure bias is directly related to the closed feedback loop in a recommendation system. In our work, we are interested in the temporal evolution of the exposed set and the missing data (i.e the unseen items).

2.2.4 Iterative or Closed Feedback Loop Bias

If we classify the bias types into a hierarchical scheme, iterative or closed feedback loop bias would subsume all the aforementioned types of bias. Depending on the initial state of the system, the closed feedback loop incorporates the user selection bias, which is itself affected by the previously seen items, suggested by the previous recommendations [17]. Hence, understanding this iterative behavior is important in order to control its effects. [12] provided a theoretical and empirical study of how the iterative process can affect the user preferences, proving that if a user is trapped in a filter bubble with no strategy to escape, the recommendation quality will decrease. [83] modeled the feedback loop through dynamically modeling the dynamic item missingness of the ratings by combining Markov Models and Matrix Factorization in order to improve the quality of the recommendations. [84] and [85] also used Markov models to model the closed loop. Other studies [18, 82] tried to empirically model the iterative bias by using simulations that study the effect of the algorithms on various diversity metrics. Furthermore, [86] tried to deconvolve the feedback loop to understand how it affects the final ratings.

Although the aforementioned studies consider the temporal dependency in recommender systems, a simplified theoretical modeling of how the system evolves has been missing from the literature until our recent work [16]. Our work falls under this category, where we consider an *iterative* collaborative filtering recommender system environment and further formulate assumptions under which the system evolves to a static state characterized by limited human discovery.

2.2.5 Why should we care about the bias in recommender systems?

The emergence of bias in recommender systems leads to several effects that range from affecting the quality of the user's experience and unfairness against items, to degrading the model's performance as shown in Figure 2.3.

As explained in Section 2.2.5, the feedback loop bias leads to echo chambers and filter bubbles. This in turn leads to the user being exposed to the same type of items

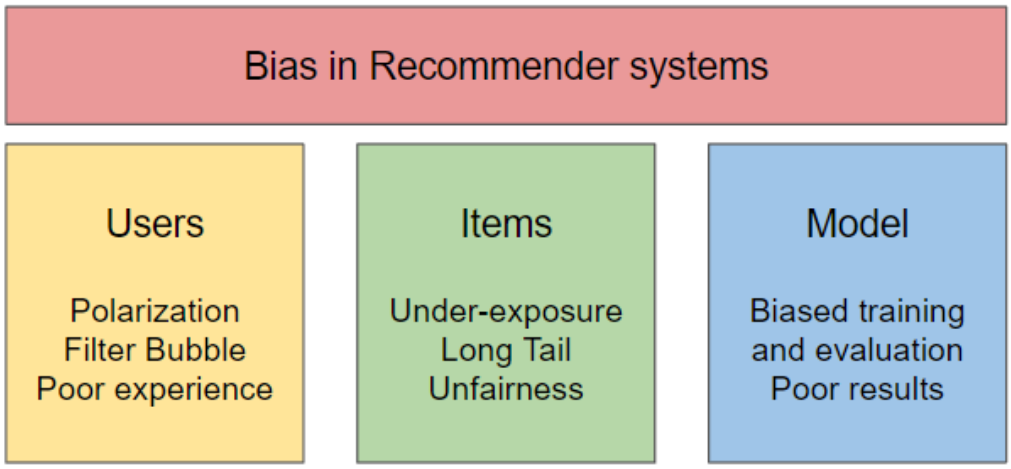


Figure 2.3. Summary of the effects of bias on recommender systems. Bias affects all the components of the recommender system, including users, items, and model.

repeatedly. Echo chambers contribute to reinforcing the users' prior beliefs as they limit their exposure to the other viewpoints. This can contribute to increase polarization between users, especially when the items are tied to controversial topics [87, 88].

It has also been shown that feedback loops and bias amplification lead to lower user utility in the long term [89]. This is due to the user being limited to a narrow set of specific items, since the recommender system is not able to generalize to more diverse items.

From the items side, feedback loops lead to the under-exposure of certain items due to the popularity bias. This disparity leads to unfairness of exposure between items and this results in a long tail of unpopular items [90].

Another important effect of bias in recommender systems is bias against under-represented demographic groups. For instance, a recent study showed that the feedback loop bias in music recommender systems leads to unfairness against female artists, since it limits their exposure compared to male artists [91].

Beyond the direct negative effects of the bias in recommender systems on users and items, bias can also affect the training and evaluation of the model, which leads to a degraded performance. For instance, and as explained in section 2.2.3, biased exposure leads to the

Missing Not At Random problem, which also creates biased performance estimation during training and testing [38]. To express this more formally, let us consider $\Delta : F \times R \to \mathbb{R}$ to be an error function that measures the performance of the recommendation model $f \in F$ over a population of ratings R. Then let us consider $\delta F \times U \times I \to \mathbb{R}$ to be an error measure that evaluates the quality of the rating prediction or ranking for a single user. The true performance of the model is given by:

$$\Delta(f, R) = E_R(\delta(f, u, i)), \tag{2.2}$$

where E is the expected value of the random variable δ over our population R. As we do not have access to the total population (most ratings are unobserved), we use estimators over the observed set of ratings in order to estimate the generalization power of our model. A common estimator used in this case is taking the average of the observed ratings:

$$\hat{\Delta}(f, R) = \frac{1}{|O|} \sum_{(u,i) \in O} \delta(f, u, i) \tag{2.3}$$

where O is the set of observed ratings. We can show that this estimator is biased, i.e the expected value of the estimator is not equal to the true estimator:

$$E(\hat{\Delta}(f, R)) \neq \Delta(f, R) \tag{2.4}$$

To prove this, we calculate the expected value of the estimator $\hat{\Delta}$:

$$E(\hat{\Delta}(f, R)) = E(\frac{1}{|O|} \sum_{(u,i) \in O} \delta(f, u, i)) \tag{2.5}$$

$$E(\hat{\Delta}(f, R)) = E(\frac{1}{|R|} \sum_{(u,i) \in R} \mathbb{1}_{(u,i) \in O} \delta(f, u, i)) \tag{2.6}$$

$$E(\hat{\Delta}(f, R)) = \frac{1}{|R|} \sum_{(u,i) \in R} E(\mathbb{1}_{(u,i) \in O}) \delta(f, u, i) \tag{2.7}$$

$$E(\hat{\Delta}(f, R)) = \frac{1}{|R|} \sum_{(u,i) \in R} P((u, i) \in O) \delta(f, u, i). \tag{2.8}$$

$P((u, i) \in O)$ is the probability that the user u has been exposed to the item i. In order to get an unbiased estimator, this probability needs to be uniform. But the biased

exposure makes this estimator biased and affects the performance estimation of the model. Without a truthful estimation, we cannot gauge how well the model will perform and therefore we cannot make reliable claims about its generalization ability.

A common way to fix this issue is by using Inverse Propensity Weighting [92], Where we estimate the user's propensity in order to debias the estimator in question. A main issue in this approach is that estimating the exposure of the user is tricky and depends on many factors like social influence, previous recommender system iterations, and item popularity. Some works have suggested using the popularity to estimate the propensity or using Poisson Factorization [82].

To summarize, we showed the importance of the impact of bias on recommender systems. This impact can affect the user directly by limiting the recommended items which leads to filter bubbles and echo chambers. This in turn can lead to more serious effects on large scale social networks. We also discussed the effect of bias on the items. We showed that bias creates unfairness between groups of items and deepens the disparity. Finally, we discussed the effect of bias on the model as it affects the training and evaluation by giving biased estimates of the performance.

In the next section, we discuss the importance of offline evaluation for recommender systems, especially to assess the bias present in these systems. We examine the simulation techniques present in the literature and we explain the main differences compared to our proposed work.

2.3 Traditional evaluation systems

Recommendation systems are dynamic by nature. Therefore, the evaluation of these systems should take into account the iterative aspect of the models. The traditional way to evaluate recommender systems in a production environment is by using A/B testing [93]. A/B testing consists of performing a randomized experiment to compare two models (or strategies), denoted as A and B. Each variant is applied to a group of users (a treatment group and a control group). The idea is to assess the significance of the difference of the

effect of the new treatment (new recommendation strategy or model). If there is a significant difference, then we can conclude, with a degree of confidence, that the new model is better.

The main issue with A/B testing is that it is expensive. In fact, in order to run a rigorous A/B testing experiment, one needs an online platform with many actual users and an experimentation framework. This is besides the cost of waiting until enough data is gathered to draw any conclusion. This in turn hinders the speed at which we can run multiple experiments to test a new model or a new component.

An alternative way to test recommender systems' performance is offline evaluation. A classic offline evaluation framework consists of performing classic train and test splitting where the data is split into training, validation and testing. Then the model is evaluated on the test data, while the validation data is used to tune the model's hyper-parameters. This approach has several flaws. First, most splitting strategies such as random splitting or stratified sampling do not reflect the real world scenario. In real life, the data comes to the system in a streaming fashion, where the time plays a major role in the arrival of data. Furthermore, most time-splitting strategy have problems because one needs to consider the date of the interaction for all the users at the same time. For instance, one cannot use future interactions of a user in order to predict the recommendations of another user, as this might leak some information from the future, and hence undermine any evaluation that aims to estimate the generalization ability of a model on new data. Some solutions have been proposed, such as sliding window techniques [93] in order to have a more rigorous way to evaluate recommendation systems in an offline fashion.

The above solutions still do not consider the dynamnic behavior of recommender systems. In fact, offline evaluation also needs to evaluate the behavior of recommender systems in a dynamic environment. For this reason, offline evaluation makes use of simulation frameworks

Several simulation frameworks have been designed by researchers for different purposes. Major work has been done on simulation frameworks for training Reinforcement Learning policies [94–98]. These frameworks aim at implementing a learning framework in

order to train a recommendation policy. Other simulations focus on modeling user behavior [99]. Recent works showed the dynamic of recommender systems feedback loop in a social network environment [100].

We differ from these systems in the *purpose* of our framework. Specifically, we include the *state model* which can be learned using a classical approach such as Matrix Factorization. We also focus on measuring the *bias* effect using our simulation instead of measuring only the accuracy. The reason for this focus is that measuring accuracy in a simulation setting is subject to correct user modeling and might also be affected by the bias coming from the different recommender system components. Some studies have worked on debiasing the simulation frameworks [94, 101]; while other studies used simulations to study the effect of recommender systems on several types of biases [12, 16, 18, 82, 102]. The framework in [103] is the closest to our work and uses a similar simulation framework to test the effect of recommender systems on popularity bias. Our approach differs from [103] as we do not consider a Reinforcement Learning setting; instead, we focus on the state model, which is kept constant in [103], i.e., without retraining. In contrast, we try to explore the impact of *retraining* the model (using an evolving dataset) on different biases, while also exploring the *interaction* between the different debiasing techniques.

The full control provided by our simulation framework allows effective recommender system evaluation and the ability to audit such systems by testing their different components for potential bias. This can be used to assess the effect of the present bias on the system. A thorough comparison of our simulation framework and other simulations is presented in Table 2.1.

2.4 Summary

In this chapter, we summarized the main literature revolving around recommender systems and bias in recommender Systems. First, we detailed the different families of recommender systems models and we explained the main architectures and models that are commonly used to achieve state-of-the-art results. Then, we reviewed research studying

TABLE 2.1. Comparison between the proposed SimBias simulator and other simulators in the literature.

Simulator	Supervised Learning	Bias Evaluation				Feedback Loop	State Model training	Selection Model	Choice Model	Feedback Model	Real Data support	Modular Support
		Diversity	Exposure	Fairness	Popularity							
RecoGym [96]	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✓
PyRecGym [97]	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗
Recsim [95]	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✗	✓
MARS-Gym [98]	✗	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓	✗
Accordion [104]	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓
SIREN [18]	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗	✗	✗
SimBias	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓

bias in recommender systems. We explained the different types of bias and then described the feedback loop bias and how it affects other types of bias. We also reviewed the existing recommender system simulation frameworks and their main limitations.

The main limitation of the previous literature studying bias in recommender systems is not accounting for the iterative behavior of such systems. In fact, most of the interest so far has been devoted to debiasing a fixed iteration of the recommender system. In our work, we propose a new theoretical framework that is helpful for modeling the recommendation feedback loop and we study the convergence behavior of the recommender system, while suggesting preliminary work on increasing the diversity of the recommendations. We also provide a simulation framework to evaluate the dynamic bias in recommender systems.

CHAPTER 3

TRANSFORMER-PROMOTER NETWORK: A SELF-ATTENTION TRANSFORMER ARCHITECTURE TO PROMOTE DIVERSE RECOMMENDATIONS

In this chapter, we propose a novel transformer-based deep learning architecture, that employs a new attention scoring mechanism to provide more diverse recommendations without compromising the predictive accuracy.

In Section 3.1, we start by explaining the additional notation and setting used we give a general overview of the proposed architecture; then we dive deeper into the technical details of the attention mechanism. Then in Section 3.2, we present our experimental results that evaluate the Transformer-Promoter Network. We compare its results with different Collaborative Filtering algorithms and test the effectiveness of the algorithm on different real-word datasets in terms of accuracy and diversity in Section 3.2.4.

The notation used in this Chapter is summarized in Table 3.1.

TABLE 3.1

Summary of the notation used for the Transformer-Promoter Network.

Symbol	Meaning	Symbol	Meaning
I	Set of items	R_{ui}	Rating of user u to item i
U	Set of users	N	Neighborhood set
Y	Embedding of the user	N_{max}	Maximum number of neighbors
Z	Embedding of the item	α	Attention weights
d	latent dimensionality	λ	Attention regularization parameter
K	Key matrix	L	Loss function
Q	Query matrix	Pop	Popularity (number of ratings)
V	Value matrix	C	Centroid embedding
R	Rating matrix	$\mathbb{1}$	Indicator function

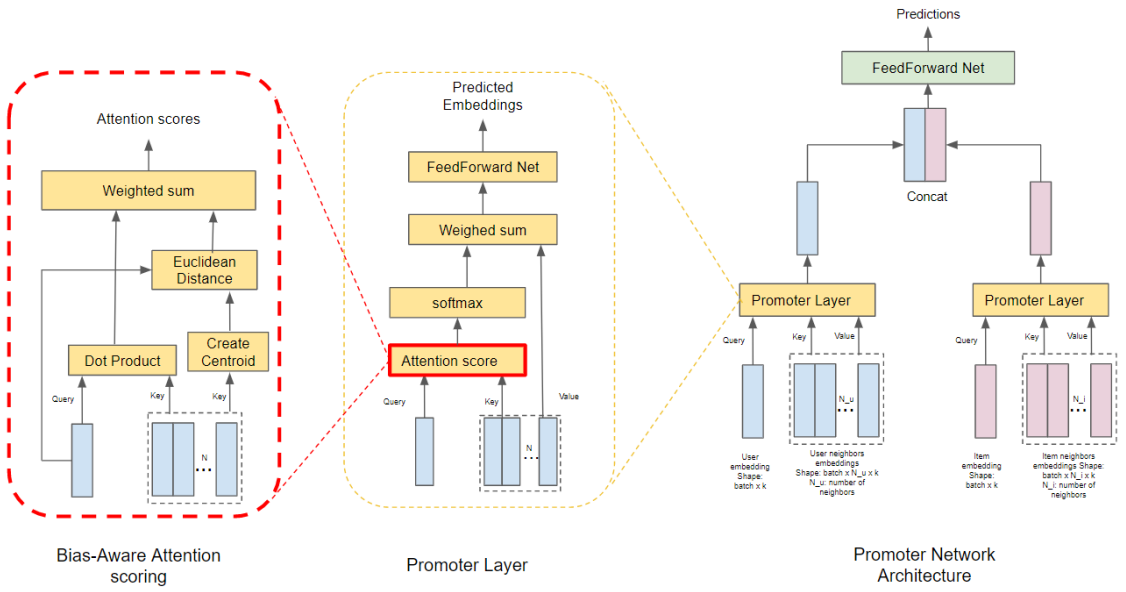


Figure 3.1. Rightmost: Overall architecture of the proposed Transformer-Promoter network. The model adopts a two-tower architecture where it jointly learns embeddings for the user u and item i. Center: Promoter layer. This is a standard Transformer layer with a modified attention mechanism. Leftmost: Bias-aware attention scoring. It uses a centroid to calculate regularization scores to the attention weights. This helps reduce the dominance of redundant elements in the neighborhood.

3.1 Transformer-Promoter Network: Methodology and Design

3.1.1 Overview and Problem Statement

We adopt a Collaborative Filtering setting for the recommendation problem. Mainly, we consider a set of users U, a set of items I, and a rating matrix R. R_{ui} denotes the rating given by user u to item i and explicitly encodes the preference of user u to item i. Although in this work, we use explicit feedback data for training the model, the proposed approach can also be extended to work with implicit feedback.

The idea of our model is to jointly learn embeddings of a certain dimension d for users and items, that we note Y and Z, respectively. Y_u designates the embedding vector for user u and Z_i designates the embedding vector for item i.

Figure 3.1 depicts the overall architecture of the model. The logic is similar to matrix factorization and other two-tower based architecture that focus on jointly learning both user

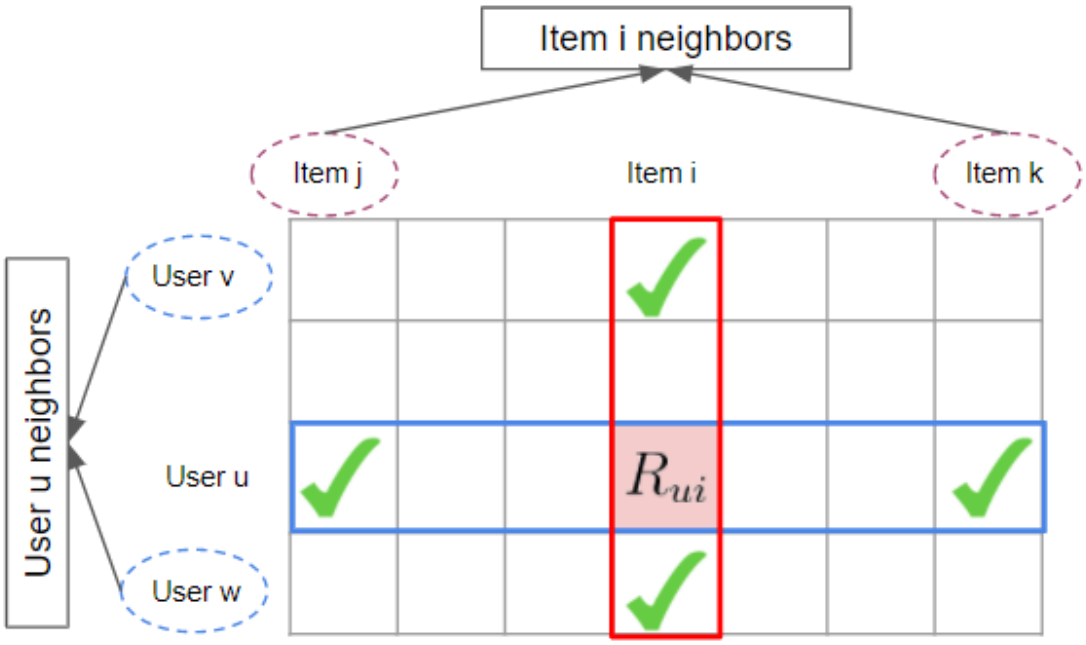


Figure 3.2. Illustration of contextual neighborhood for User u and Item i, In this scenario, since user u has rated items k and j then the neighborhood of item i is $\{item_j, item_k\}$) and since item i has been rated by user v and w then the neighborhood of user u is $\{user_v, user_w\}$)

and item embedding. The inputs are the user and item embeddings and the embeddings of their neighbors (which will be explained in the next section). These embeddings are fed to a Promoter Layer. The Promoter layer acts like the Transformer Layer but uses a different Attention mechanism. In fact, we create attention weights by attending not only to the closest neighbors of the query but also to less similar neighbors by regularizing the attention score. The reason for this regularization is that the classic similarity scoring method is biased due to the feedback loop.

After creating the new embedding for the users and items, we concatenate them and use a feedforward network for the final prediction.

3.1.2 Promoter Network

3.1.2.1 Inputs: Query, Key, and Value

We adopt the same notation as the Transformer architecture by using *Query*, *Key*, and *Value* as inputs to our network. However, in the Transformer Network, Query, Key, Query Value all denote a different representation of the same sequence. In our case, Query designates the user or the item embedding for which we want to predict the interaction. Key and Value represent different representations of the neighborhood of the user and the item.

The key aspect of the input for the Transformer-Promoter Network is defining the neighborhood of the user and the item. To get these neighborhoods, we create contextual neighbors. This is shown in Figure 3.2. In fact, for a given context (u,i), the neighbors of user u are the users that rated item i, and the neighbors of item i are the items that have been rated by user u. The intuition is that for each context (u,i), the neighbors of user u and i should change. This is analogous to word-positional neighborhoods in text, where the neighbors of each word change depending on the context.

Using this approach, we get different entities affecting the predicted embedding with each context. Therefore the learned embeddings are *contextual* where they change depending on the context provided, which in this case, is a user u and item i.

3.1.2.2 Promoter Layer

The Promoter layer is very similar to the Transformer layer, but using a different attention mechanism. We will describe the main components for the promoter layer, then explain the attention mechanism in more detail. Using the Query and Key defined in the previous section, we first calculate the attention scores. Assuming f is a scoring function where $f : \mathbb{R}^d \times \mathbb{R}^{|N| \times d} \rightarrow \mathbb{R}^{|N|}$, we compute the attention weights as follows:

$$\alpha_j = softmax(f(Q, K)) \tag{3.1}$$

where α_j is the attention weight designating the contribution of the neighbor j to the final output. After computing the attention weights, the final embedding is:

$$output = Feedforward(\sum_{j \in N} \alpha_j V_j) \tag{3.2}$$

with Feedforward in our case being a Relu function followed by a linear transformation with a residual connection.

$$Feedforward(x) = x + W^T.Relu(x) \tag{3.3}$$

We sum over all elements of the Value matrix while weighting by their attention score. The resulting embedding is therefore an aggregation of the embeddings of the neighborhood. This method has the advantage of explicitly forcing the collaborative filtering paradigm within the training. In fact, in classical Collaborative Filtering methods, we only assume that the similar entities (users or items) will have similar embeddings, but this is not explicitly provided in the model architecture. Thus, the Transformer-Promoter Network has the advantage of using the *neighborhoods* of both users and items in order to force this collaborative behavior.

3.1.2.3 Bias-Aware Attention

Classical attention scoring mechanisms revolved around using dot products (or scaled dot products) to capture the most similar key elements to the given query. This assumes that the keys are selected from an unbiased setting and can fully represent the query neighborhood. In the case of recommender systems, the collected data is biased and the neighbors of each item or user might be dominated by redundant elements such as a popular type of items or a similar groups of users.

For instance, remembering the example in Figure 1.1, in order for the item's neighborhood to be unbiased given a context (u,i), the item's neighborhood N_i should contain an equal number of comedy and action movies to fully represent the user taste. To further show the inefficiency of previous attention techniques, we assume that the scoring function

is a simple dot product as shown in Equation 3.4.

$$\alpha_j = softmax(\frac{Q^T.K}{d}) \tag{3.4}$$

Now we assume that the query has equal cosine distance to all the elements of the key as depicted in Figure 1.1. The classic attention approach will provide equal weights to all Key elements: $\alpha_j = \alpha_i \ \ \forall i, j \in N$. In this case, when we compute the resulting embedding from V, the sum $\sum_{j \in N} \alpha_j V_j$ will give higher weights to the action items' embeddings since they are redundant in the Key. Hence the resulting embedding will be influenced by this redundancy; and the comedy movies, even though relevant, will not be fairly accounted for in the prediction.

Solving this problem would have been easy if we had labels where we can estimate the redundancy of each item. However, since we are relying only on the embeddings, we need to adopt an unsupervised approach. Our idea is to represent the neighborhood by a centroid C that is calculated from all the elements of the Key. In fact, this centroid will capture the majority class as its embedding will be closer to the redundant elements. Hence, the more an item is distant from the centroid, the rarer it is in the neighborhood. Using this intuition, we regularize the score of each element in the key by its distance to the centroid. Therefore the new attention score becomes

$$\alpha_j = softmax(\lambda \frac{Q^T.K}{d} + (1 - \lambda)||K - C||_2) \tag{3.5}$$

$$C = \frac{1}{N} \sum_{j \in N} K_j \tag{3.6}$$

The coefficient λ controls the regularization within the attention function. $\lambda = 1$ corresponds to the classical scaled dot product scoring function.

The calculated attention scores will up-weight the scores for elements that are far (in terms of $L2$ norm) from the centroid and therefore not similar to most of the present elements. This compensates for the redundancy of the majority class and hence promotes diverse and less common recommendations, while keeping powerful and accurate item and

Algorithm 3.1 Prediction steps for Promoter Layer for items

Input: Item Id i Item Neighbors ids N_i
Output: Predicted Embeddings $\hat{Z}_i$ for item i
Create Query: $Q \leftarrow Z_i$
Create Key: $K \leftarrow Z_{N_i}$
Create Value: $V \leftarrow Z_{N_i}$
Calculate Centroid embedding:
$C \leftarrow \frac{1}{N} \sum_{j \in N_i} K_j$
Calculate attention scores
$\alpha_j \leftarrow softmax(\lambda \frac{Q^T.K}{d} + (1-\lambda)||K - C||)$
Get predicted embedding
$\hat{V}_i \leftarrow Feedforward(\sum_{j \in N} \alpha_j V_j)$
Return $\hat{V}_i$

user representations. The algorithm for the Promoter Layer embedding prediction for the items is listed in Algorithm 3.1. The same procedure is applied to the users but using the user embeddings instead of the item embeddings.

The Bias-Aware attention mechanism, when applied to the users, is expected to help reduce the filter bubble effect and promote novel recommendations. In fact, users differ in their tendency to rate items. Some users may rate many more items than others. Therefore, some user neighborhoods will be affected by the users who always rate items and may not reflect the true similar preferences that these neighbors share. For instance, a user who might like both horror and romance movies might end up with their neighborhood being biased toward one genre more than the other. The Bias-Aware attention function helps to capture this disparity and promotes novel recommendations to increase the user exposure to more items.

3.1.2.4 Complexity Analysis

The complexity of our network is similar to the Transformer's layer complexity [30]. In fact we have a quadratic time complexity in terms of number of neighbors and depends on the latent dimension d $O(N_i^2 d)$. The regularization is linear in terms of the number of neighbors N_i:$O(N_i + d)$. Therefore, in order to optimize the time complexity, we aim at optimizing the self attention unit. For future work, one can apply several optimization

techniques to reduce this complexity further using same techniques proposed by [105].

3.2 Evaluation of the Transformer-Promoter Network

In this section, we evaluate the empirical performance of the Transformer-Promoter model by answering the following research questions:

- **RQ1.1:** How does the Transformer-Promoter Network affect the diversity and novelty of the recommendations?
- **RQ1.2:** How accurate are the recommendation provided by the Transformer-Promoter Network?
- **RQ1.3:** How does the Bias-Aware Attention unit affect the bias and accuracy?

In the following, we describe the datasets used in the experiments, the state of the art models that we used to compare with, and the implementation details.

3.2.1 Datasets

We use three publicly available datasets in order to evaluate the performance of our model from different domains. Table 3.2 shows the different statistics of the used datasets.

- **Movielens-100K**[1]**:** A movie rating dataset with ratings from 1 to 5. The data set has 95% sparsity.
- **Movielens-1M**[2]**:** A larger version of the Movielens 100K. We use it to evaluate the scalability of the model on larger datasets.
- **Epinion**[3]**:** The dataset contains ratings from 1 to 5 given by users to different items in a shopping platform. It has a very high sparsity: 99.99 %. We use it to evaluate the performance of our algorithm on a different recommendation domain.

[1] `https://grouplens.org/datasets/movielens/100k/`
[2] `https://grouplens.org/datasets/movielens/1m/`
[3] `https://snap.stanford.edu/data/soc-Epinions1.html`

In order to get meaningful neighborhoods for our model, from each dataset we sample users and items that have at least 20 interactions. This allows us to reduce the sparsity and get better embeddings.

3.2.2 Baselines

We compare to three Collaborative Filtering algorithms: Matrix Factorization (MF) [8], Neural Matrix Factorization (Neumf) [9], and ItemPop [106], summarized below.

- **Matrix factorization:** Matrix factorization is a popular CF technique that learns an embedding of users and items by decomposing the rating matrix into two embedding vectors. It uses the dot product to map the embedding to predicted ratings.
- **NeuMF:** Neumf combines Matrix Factorization decomposition techniques with MLP layers in order to learn higher-level representations for users and items. Neumf is one of the first CF neural networks based models.
- **ItemPop:** ItemPop is a non-personalized recommendation method that recommends to the user the most popular item. We use it as a benchmark for performance.

Each model was tuned separately to get the best parameters for a better accuracy.

3.2.3 Implementation Details

3.2.3.1 Neighborhood size $|N|$

$|N|$ is the number of neighbors that item i or user u can have. The number of neighbors can be either very large for very popular items and very active users, or very small for nonpopular items and users that tend to not rate many items. This inequality poses a constraint for deep learning models as it creates unequal batch sizes. In fact, in order for our model to be scalable and to be used on GPUs, all the items and users must have an equal number of neighbors. For this reason, we use a padding index in the learned embedding and we use this index to augment the tensors to a length equal to N_{max} which

designates the maximum number of neighbors that a user or item can have. Furthermore, since recommender systems can face a large number of users and items, we set a condition on $|N|$ which designates the number of neighbors (for users or items), to be bounded by N_{max}. This is achieved by randomly sampling N_{max} neighbors if the neighborhood size exceeds N_{max}. This allows systems with a very large number of users and items to have a reasonable and controllable size of tensors. In addition, sampling increases the diversity of the neighborhood in large systems where the number of neighbors is usually high. In our experiments, we set $N_{max} = 512$. This can be changed according to the size of the problem.

3.2.3.2 Splitting and Tuning

We adopted an 80-10-10 random splitting scheme. We train on 80% of the total training data, we use 10% for validation in order to tune the hyperparameters and final 10% for testing.

Hyperparameter tuning for all baselines was done using Optuna employing Tree-structured Parzen Estimator [107] as a sampler. We tune on the Movielens 100K and we keep the same hyperparameters for the other datasets. The reason for this is to avoid overfitting the validation data and test the model on new ratings. The used hyperparamters are represented in Table 3.3

3.2.4 Evaluation

We evaluate the performance of our proposed Transformer-Promoter model against two main criteria: Bias and Accuracy.

3.2.4.1 Bias evaluation metrics

Bias in recommender systems can be related to different aspects [70] such as statistical bias, feedback loop, exposure bias, etc. In this work, we evaluate three main aspects of the bias in recommender systems: Diversity, novelty, and popularity. We mainly want to have diverse recommendation lists that have new items (different from the previously

seen ones) and with low popularity for more exposure. To test the popularity bias, we use the Average Popularity of the recommended list and Gini index on the distribution of the number of ratings for each recommended item.

$$Avg\_Pop = \frac{1}{|Rec|} \sum_{i \in Rec} Pop_i \tag{3.7}$$

$$Gini(x_i) = \frac{\sum_i (2i - n - 1)x_i}{n \sum_i x_i} \tag{3.8}$$

Furthermore, we test the diversity of the recommendation lists using Intra List Diversity (ILD) [108].

$$ILD = \frac{2}{|Rec|(|Rec| - 1)} \sum_{i,j \in Rec, i \neq j} 1 - cosine\_similarity(i, j) \tag{3.9}$$

Finally, we test the novelty of the recommendations using the Novelty metric [109]

$$Novelty = \min_{i,j \in Rec, i \neq j} 1 - cosine\_similarity(i, j) \tag{3.10}$$

To assess the similarity between two items, we use the cosine similarity between their rating vectors.

3.2.4.2 Accuracy evaluation metrics

Accurate recommendations are crucial for retaining the user interest. We evaluate two accuracy criteria: Accuracy of rating prediction and accuracy of ranking. For the rating prediction, we use Root Mean Squared Error (RMSE). To assess the quality of the recommendation list, we use Precision@5, Recall@5, and NDCG@5. We choose to use only the top 5 in the case of testing the accuracy to test the accuracy of the highly ranked items. To prevent the user's personal bias from affecting the ranking metrics (two users giving a 4 rating to an item might mean different preferences because of their biases), we only consider an item to be relevant if it has a rating of 5.

3.2.4.3 Evaluation protocol

To evaluate the diversity performance of the models, we predict the rating of the user to all the items that do not have ratings and then we calculate the bias metrics on the top 10 recommended list (based on the predicted score). We do not need ground truth for the diversity bias because the metrics evaluate the diversity of the list regardless of the true relevance. However, to evaluate the accuracy, we need the ground truth: true ratings in the case of RMSE and true ranking in the case of ranking metrics. For this task, we only predict the ratings for the items in the test set (items that have ratings). In this scenario, we test the following hypothesis: If $R_{ui} > R_{uj}$ then $\hat{R}_{ui} > \hat{R}_{uj}$. All experiments were repeated 5 times to account for randomness. The average of these runs and standard deviation are reported.

3.2.5 Experimental Results

3.2.5.1 RQ1.1: How does the Transformer-Promoter Network affect the diversity and novelty of the recommendations?

Table 3.4 shows the diversity, novelty, and popularity on the three datasets. We observe that the Transformer-Promoter Network significantly outperforms the other models in terms of Diversity (ILD) and Novelty across all datasets. It scores 0.7 on the Movielens 100K, 0.76 on the Movielens 1M, and 0.97 on the Epinion dataset. A high ILD metric indicates that the Transformer-Promoter Network is providing diverse recommendation lists based on the pairwise distances between these items. Furthermore, the Transformer-Promoter Network significantly outperforms other models in terms of Novelty for the Movielens data and has comparable result to NeuMF+MAB on the epinion dataset. This shows that the model is able to recommend new items to the user. Matrix Factorization and NeuMF have comparable results in terms of bias and Novelty, while Item-Pop has very biased recommendations, as expected, since the recommendation strategy used is biased toward popular items.

We also notice that Transformer-Promoter Net is able to provide less popular items

based on the Average Popularity score. This shows that the model achieves better coverage by increasing the exposure of unpopular items. Although MF and NeuMF achieve better coverage than Item-Pop, they remain biased towards popular items. We observe that our model has a high Gini Index relative to the other models, this shows that the disparity between the popularity in the recommendation list is higher. This is due mainly to the fact that MF, NeuMF, and Item-Pop recommend more popular items, and hence, the difference in terms of popularity between the recommended items is low.

3.2.5.2 RQ1.2: How accurate are the recommendations provided by the Transformer-Promoter Network?

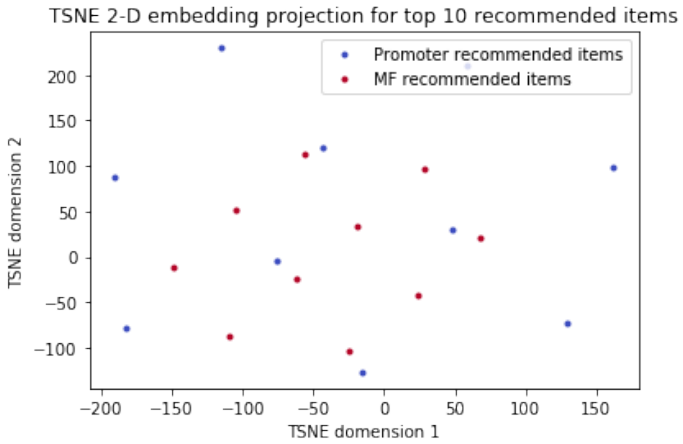
Table 3.5 shows the results in terms of rating prediction accuracy and ranking accuracy. In terms of rating prediction, Transformer-Promoter Net has a comparable RMSE to MF and NeuMF for the Movielens 100K and Epinion datasets. However, in the case of the Movielens 1M, Transformer-Promoter Net significantly outperforms other baselines by having 0.90 RMSE. This indicates that the Transformer-Promoter Network scales better with more data. Rating prediction is important to evaluate how a user will like a given item, however, ranking is also crucial for more relevant information filtering. Our results show that in terms of precision and recall, the model performs similar to MF and NeuMF, with no significant difference in the results. Furthermore, the NDCG metric shows on par performance of our model with state of the art baselines. This proves that the Transformer-Promoter Network is pushing more relevant recommendations to the top of the recommended list, even while diversifying the lists.

3.2.5.3 Bias and Accuracy Trade-off

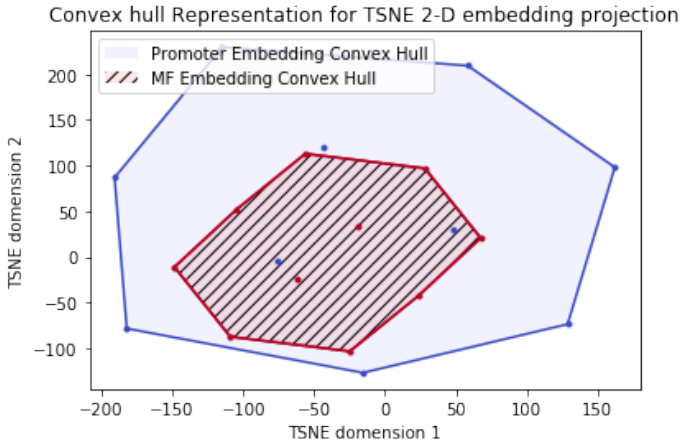
With more exploration, the accuracy of the recommendation model decreases [110]. This is especially true for post-processing methods that introduce noise to the predictions to increase the exploration [111]. In contrast, our results show that the Transformer-Promoter does not compromise its accuracy in order to increase the diversity of the recommendations.

The main reason for this is incorporating the debiasing *within the training* process. In fact, the key idea in our methodology is learning diversified *embeddings*, meaning that the embedding of the items and users that we are learning are regularized in a way to promote diversity, while *also* capturing a good representation of the user-item interactions.

3.2.6 Learning Diverse Embeddings



(a) Projection of the top 10 items in a two dimensional space using TSNE.



(b) Convex hull for the projection of the top 10 recommended items.

Figure 3.3. **Projection of the top 10 items for a given user in Movielens-100k in a 2D space. Each point in Figure 3(a) represents the projection of the embedding of one item recommended to the user. Figure 3(b) shows that the recommended list for MF is more compact and hence it has less coverage. On the other hand, we observe a larger hull for the Transformer-Promoter Network showing a more diverse embedding**

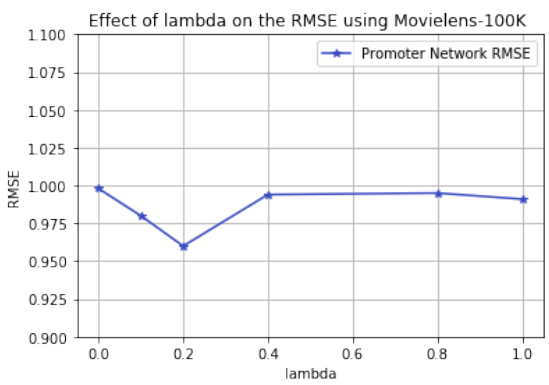
To further understand the key contribution of including the diversity constraint within the learning of the embedding, Figure 3(a) shows the projection of a recommendation list for a given user in the Movielens-100K dataset for both Transformer-Promoter Network and Matrix Factorization. The projection was performed using the t-distributed stochastic neighbor embedding method (t-SNE) [112]. The idea is to notice how far away the embeddings are from each other and how compact the item embeddings are in the recommendation list. To further understand this, we plot the convex hull for each recommendation list for both Transformer-Promoter and MF in Figure 3(b). The figure shows that the recommendation list provided by MF is more compact and the items are more similar to each other. In contrast, the Transformer-Promoter Network results in a bigger convex hull which shows a higher coverage and more diversity in the embeddings of the recommended items. The intuition for this result is the use of the distance to the centroid of the Key K as a regularization parameter. In fact, using this distance pushes the embedding to be far from the center of its neighborhood.

We emphasize that the goal of our model is to learn a **diverse embedding** rather than simply providing diverse recommendations, as will be shown in our experimental results. The key distinction is that by learning a diverse embedding, we *also* maintain an *accurate* representation of the user-item interactions and therefore do not compromise accurate recommendations.

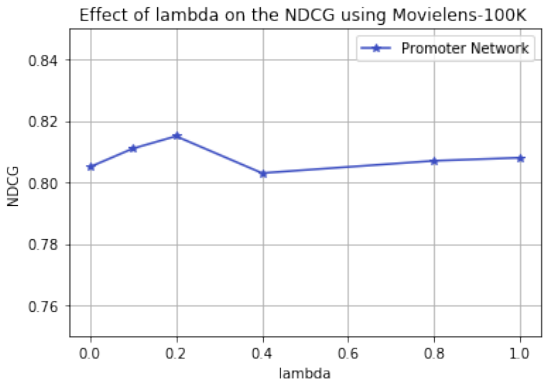
3.2.7 Ablation Study

3.2.7.1 RQ1.3: How does the Bias-Aware Attention unit affect the bias and accuracy?

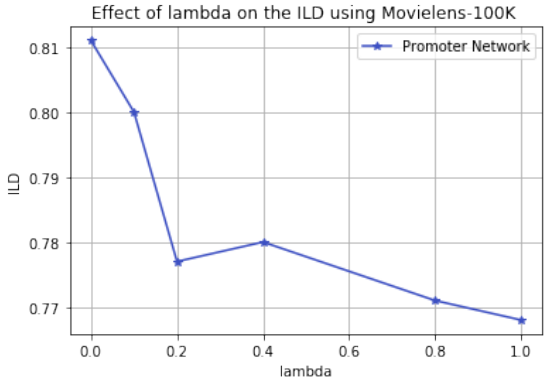
In order to assess the effectiveness of our new Attention mechanism, we compare the performance of our Bias-Aware Attention unit to the scaled dot-product attention. The idea is to evaluate the contribution of our attention mechanism to learning diverse recommendations. Table 3.7 compares the results of our Attention mechanism with the Scaled Dot Product mechanism, adopted in many attention-based models. Both models



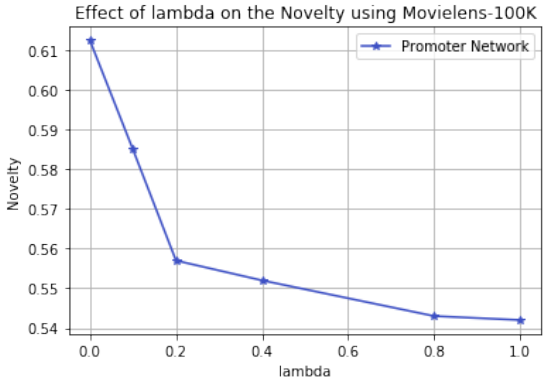
(a) Effect of λ on the RMSE



(b) Effect of λ on the NDCG.



(c) Effect of λ on the ILD.



(d) Effect of λ on the Novelty.

Figure 3.4. **Effect of the controlling parameter λ on the different evaluation metrics. We observe that λ has no significant effect on the accuracy of the recommendations although it does increase the diversity and novelty of the recommendations**

achieve comparable accuracy results. This observation shows that our modified attention mechanism is not sacrificing the accuracy of the model. Table 3.7 also shows the effect of the Bias Aware Attention mechanism on the diversity bias. The results show that it improves the diversity of the recommendations as it provides a significant improvement in the ILD metric. Furthermore, we see a significant improvement in the Novelty metric which shows that the new Attention mechanism promotes new items. Finally, we see that the Bias-Aware Attention unit promotes more underexposed items, as it has more unpopular items in its recommendations.

Finally, we plot the RMSE, NDCG, ILD, and Novelty against different values of λ which controls the attention weights regularization, as shown in Figure 3.4. We observe

that λ does indeed increase the diversity and novelty of the recommendations. In fact, with $\lambda = 1$, meaning that we are using the plain scaled dot product attention, we observe that the ILD and Novelty have decreased compared to $\lambda = 0$ which refers to using the L_2 norm as the attention scoring and we ignore the key-query similarity. What is surprising about these results is the fact that the accuracy was not affected by varying λ. The reason for this behaviour is the way we are constructing the neighbors i.e. the Key K. In fact, by selecting the neighbors for the items and the users, we are already including a bias toward the interacted item, and the learned embedding will be close to the embedding of the key that was used in this case. Hence, even without using the Query projection in the network, the accuracy was not affected.

3.3 Summary

In this chapter, we proposed a new recommendation model that we called the *Promoter Network*, a transformer-based deep learning architecture using a novel attention framework in order to improve the diversity of the recommendations, and thus alleviate the problem of bias. The Transformer-Promoter network uses a new self-attention framework that regularizes the attention weights in order to focus more on the under-represented items and therefore improves the generalization of the learned embeddings.

We then evaluated the performance of the proposed Transformer-Promoter network in terms of the accuracy and diversity of the recommendations by using several real-world datasets. We notice that Transformer-Promoter network learns a diverse set of embeddings for both users and items while keeping accurate results.

TABLE 3.2

Statistics of the used datasets

Dataset	#Users	#Items	#Interactions	Sparsity
Movielens-100k	943	1682	100 000	93.6%
Movielens-1M	6040	3883	1M	95.7%
Epinion	49 289	139 738	664 824	99.99%

TABLE 3.3

Hyperparameters used for training the Transformer-Promoter Network

Learning rate	Latent dimension	Feedforward Dim	λ	optimizer
0.046	16	64	0.9	Adagrad

TABLE 3.4. Performance of diversity, novelty, and popularity bias in the recommendations with three different datasets. Significant results are in bold ($p-value < 0.05$) and comparable results are underlined. We observe that the Transformer-Promoter Net significantly outperforms all the other models.

Dataset	Movielens-100K				Movielens-1M				Epinion			
Metric	ILD	Novelty	Avg_Pop	Gini	ILD	Novelty	Avg_Pop	Gini	ILD	Novelty	Avg_Pop	Gini
MF	0.6 ± 0.015	0.35 ± 0.012	0.26 ± 0.01	0.19 ± 0.007	0.51 ± 0.002	0.344 ± 0.004	0.35 0.002	0.08 ± 0.0002	0.92 ± 0.006	0.83 ± 0.008	0.04 ± 0.001	0.13 ± 0.01
NeuMF	0.67 ± 0.03	0.42 ± 0.018	0.17 ± 0.02	0.27 ± 0.03	0.66 ± 0.018	0.44 ± 0.02	0.198 ± 0.012	0.21 ± 0.022	0.96 ± 0.013	0.85 ± 0.04	0.011 ± 0.0039	0.32 ± 0.034
Item_Pop	0.51	0.25	0.44	**0.03**	0.47	0.26	0.39	0.03	0.82	0.74	0.13	**0.11**
MF + MAB	0.6 ± 0.02	0.4 ± 0.02	0.17 ± 0.02	0.264 ± 0.007	0.52 ± 0.004	0.378 ± 0.003	0.35 ± 0.005	0.07 ± 0.002	0.93 ± 0.003	0.86 ± 0.01	0.05 ± 0.001	0.11 ± 0.01
NeuMF + MAB	0.654 ± 0.011	0.43 ± 0.001	0.185 ± 0.02	0.24 ± 0.01	0.655 ± 0.01	0.48 ± 0.01	0.19 ± 0.025	0.24 ± 0.01	0.96 ± 0.02	**0.9 ± 0.05**	0.01 ± 0.001	0.34 ± 0.05
Promoter	**0.7 ± 0.01**	**0.44 ± 0.02**	0.18 ± 0.01	0.26 ± 0.02	**0.766 ± 0.01**	**0.53 ± 0.018**	**0.11 ± 0.017**	**0.36 ± 0.03**	**0.97 ± 0.005**	0.894 ± 0.01	0.010 ±0.001	0.34 ± 0.026

TABLE 3.5. Performance of predicting accurate recommendations with three different datasets. Significant results are in bold (p-value ¡0.05), comparable results are underlined. We observe that the Transformer-Promoter Net has comparable results to the state of the art while having better rating prediction accuracy in the case of Movielens-1m

Dataset	Movielens-100K				Movielens-1M				Epinion			
Metric	P@5	Recall@5	NDCG@5	RMSE	P@5	Recall@5	NDCG@5	RMSE	P@5	Recall@5	NDCG@5	RMSE
MF	**0.33 ± 0.01**	.6 ± 0.01	.8 ± 0.003	**0.924 ± 0.007**	0.39 ± 0.0023	0.43 ± 0.001	0.77 ± 0.0008	1.15 ± 0.006	0.31 ± 0.002	0.87 ± 0.0027	0.97 ± 0.0007	1.175 ± 0.005
NeuMF	0.32 ± 0.0085	.591 ± 0.012	.81 ± 0.002	0.932 ± 0.012	00.41 ± 0.003	0.45 ± 0.002	0.75 ±0.001	0.93 ± 0.005	0.32 ± 0.003	0.88 ± 0.003	0.97 ± 0.005	1.068 ± 0.005
Item_Pop	0.28	0.51	0.8	NA	0.34	0.39	0.75	NA	0.26	0.79	0.96	NA
Promoter	0.327 ± 0.008	.6 ± 0.02	.813 ± 0.006	0.94 ± 0.009	0.41 ± 0.004	0.45 ± 0.003	0.75 ± 0.003	**0.90 ± 0.002**	0.32 ± 0.001	0.88 ± 0.006	**0.977 ± 0.0004**	1.062 ± 0.012

TABLE 3.6

Comparison between Scaled Dot Product Attention and Bias-Aware Attention in terms of accuracy

Model	NDCG@5	P@5	R@5
Promoter +Bias-Aware Attention	0.8 ± 0.006	0.32 ± 0.008	0.6 ± 0.02
Promoter + Scaled Dot Product Attention	0.81 ± 0.003	0.336 ± 0.003	0.601 ± 0.02

TABLE 3.7

Comparison between Scaled Dot Product Attention and Bias-Aware Attention in terms of diversity and bias

Model	ILD	Novelty	Avg_Pop
Promoter +Bias-Aware Attention	**0.7 ± 0.01**	**0.44 ± 0.02**	**0.18 ± 0.01**
Promoter + Scaled Dot Product Attention	0.65 ± 0.02	0.41 ± 0.003	0.21 ± 0.002

CHAPTER 4

THEORETICAL MODELING OF USER DISCOVERY IN A FEEDBACK LOOP ENVIRONMENT

In this Chapter, we present a theoretical study of the user discovery in a feedback loop setting. The purpose is to motivate the feedback loop problem by studying its effect on the exposure of the user.

In Section 4.1, we start by presenting a new theoretical model of the recommmender system feedback loop, where we are interested in modeling the evolution of user discovery through the consecutive iterations of a recommender system. Then in Section 4.2, we empirically validate the new theoretical findings that we have developed in Chapter 3. We first perform hypothesis testing to confirm the validity of our assumptions in Section 4.2.1. Then we test the correctness of our derived lower bound on the user discovery in Section 4.2.2.2. Finally, in Section 4.2.3, we relax the assumptions that we have made in our theoretical proofs, in order to evaluate the validity of our results beyond the theoretical constraints.

4.1 Theoretical Modeling of the feedback loop in a Collaborative Filtering System

4.1.1 Notation

We start by defining the notation we will use throughout the paper that we also summarize in Table 4.1. A recommender system in a collaborative filtering setting can be fully defined by the quadruple (I, U, f_t, t) where:

- I represents the set of all available items.

TABLE 4.1. Summary of the notation used for the theoretical feedback loop modeling

Symbol	Meaning
I	Set of all items
U	Set of Users
f_t	Recommender selection function
t	iteration
g	a group of items ($g \in G$)
h_p	Ranking function related to measure P_f
G	Set of item groups
B_t	Blind spot of the user up to iteration t
Rec_t	recommended groups of items at iteration t
ΔS_t	User discovery
Rel	Set of relevant groups of items
ΔB_t	Evolution of the blind spot
S_t	Seen item groups up to iteration t
$\|.\|$	Cardinality
$\mathbb{1}_u$	Indicator function of condition u
P_f	Probability that an item from a given group will be recommended
O	Ordering score provided by P
M	Mapping function that maps items to groups

- U is the set of users. [1]
- $f_t : G \to I$ is the selection function of the recommender system which selects a fixed number of items to recommend to a given user at iteration t.
- t is the iteration number in the recommendation process as we are interested in the sequential dependency of the system.

In this work, we are interested in studying the user discovery behaviour and therefore we study the interaction of "groups" of items, as was done by [12], rather than "single" elements. A group represents a neighborhood of similar items based on some criteria such as genre, type, or nature of items. We therefore define G as the set of item groups. This will be useful later to define the user discovery capacity and the blind spot. In fact, items that are from the same group of items (share the same type, genre or characteristics, etc) as items that have been seen before may not be considered to be significantly new and may be considered to not contribute to increase human discovery, as much as items from totally

[1] Users in this work can refer to groups or neighborhoods of users (users that share the same characteristics based on a given distance measure).

unseen groups. We also define M: $I \rightarrow G$, a mapping function that maps the items to a set of item groups.

We further define Rec_t as the set of item groups that are recommended to a user at iteration t and Rel as the set of item groups that are relevant to a given user.

We also define the set S_t which represents the set of item groups that have been seen up to iteration t. As this set influences the *discovery capacity* of the user, we are interested in its evolution throughout the iterations. We illustrate the process of the iterative recommendation setting studied in **Figure 5.2**.

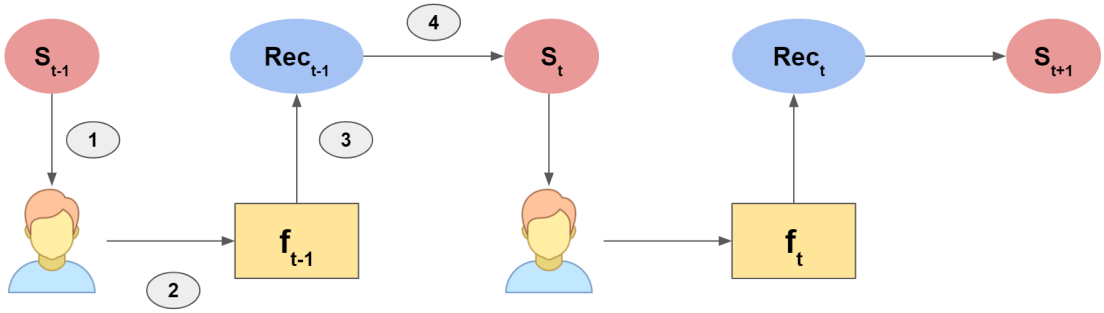


Figure 4.1. Illustrative example for an iterative recommendation process. **Step 1** represents the event of the user seeing some item groups S_{t-1}; then in **Step 2**, the user provides ratings to the seen items. In **Step 3**, the ratings are used by the recommender system function f_{t-1} in order to provide recommendations represented by Rec_{t-1}. These recommendations are finally added in **Step 4** to the previously seen item sets to form S_t. The process continues for several iterations using the same steps.

Finally, we define $B_t = S_t^c \cap Rel$ as the *blind spot* of the user at iteration t. The blind spot in a recommender system defines the item groups that have not been seen by the user, *although they are relevant.*

In practice, we want to increase the user's discovery ability (by increasing the size of S_t), but without recommending irrelevant groups. This goal translates to decreasing the size of the blind spot B_t.

We are interested in studying the evolution of S_t and B_t throughout increasing iterations. For this reason, we define:

- $\Delta S_t = |S_t| - |S_{t-1}|$: The increase in the number of new groups of items between consecutive iterations.
- $\Delta B_t = |B_{t-1}| - |B_t|$: The decrease in the number of unseen and relevant groups of items between consecutive iterations.
- $\sum_t \Delta S_t$ is the user's *discovery capacity* and $\Delta_n S = \frac{1}{n} \sum_{t=0}^{n} \Delta S_t$ is the human average discovery.

In the next sections, we study the asymptotic behavior of $\Delta_n S$ and $\Delta_n B$ and show that under certain assumptions, $\Delta_n S$ decreases exponentially with the iterations.

4.1.2 Assumptions

We make three assumptions about the nature of the iterative recommendation process, that will allow us to study the interaction between the user and the recommender system, while being isolated from any extraneous factors. The assumptions will be later tested empirically and further challenged by relaxing the ranking requirement by adding random exploration.

Assumption 4.1.1 (Vacuum Assumption)**.** *Let (I, U, f_t, t) be a recommender system, S_t is the set of seen item groups and Rec_t is the set of newly recommended item groups at iteration t. We assume that:*

$$S_{t+1} = S_t \cup Rec_t \tag{4.1}$$

This assumption is necessary to allow us to study the impact of the recommender system in isolation of any other external factors. Although it would be interesting to study the impact of extraneous factors that could lead to the user seeing an item without interacting with the seen items, we believe that this would be premature at this stage. These additional factors could include display bias and user discovery occurring outside the recommender system itself.

Assumption 4.1.2 (Perfect Feedback Assumption)**.** *Let (I, U, f_t, t) be a recommender system and let M be the item-to-group mapping function, S_{t-1} be the set of seen groups of*

items, and Rec_t be the set of newly recommended groups of items at iteration t. We assume that:

$$Rec_t = M \circ f_t(S_{t-1}) \tag{4.2}$$

This assumption simply states that the recommender system provides the next set of recommendations using the information provided by S_t. This is the case for a collaborative filtering recommender system that relies on feedback in the form of ratings from the user on the seen item groups. This assumption also eliminates the possibility of the user seeing an item without rating it.

Assumption 4.1.3 (Ranking Assumption)**.** *Let $g_i, g_j \in G$ be two groups of items and let $d : G \times G \to \mathbb{R}$ be a distance measure between two sets in G. If $d(g_i, S_{t-1}) \leq d(g_j, S_{t-1})$, then we have $P_f(g_j \in Rel) \leq P_f(g_i \in Rel)$*

What this assumption states is that a group of items has a higher probability to be recommended if it has been seen before. In other words, the recommender system maximizes item relevance by ranking the items closest to those that have already been seen, and relevant groups of items, higher than other items.

This is probably the strongest assumption we make but it has strong practical justifications. The assumption suggests that the recommendation strategy is purely exploitation-based and excludes strategies based on exploration such as Multi-Armed-Bandits methods. Most collaborative filtering recommender systems such as Matrix Factorization and Nearest Neighbors Methods, which are some of the most popular methods, satisfy this assumption, as they are trying to minimize a distance, embedded in the objective function, between the unseen items and the seen items. In Section 4.2.1 we will present an empirical validation of this assumption. Then in Section 4.2.3 we will challenge this assumption by adding an exploration component.

4.1.3 Asymptotic behavior of human discovery

It should not come as a surprise that a recommender system will eventually limit the variety of new items provided to the user. Our aim however is to provide a *theoretical*

framework to estimate the speed of convergence of this discovery limitation and hopefully lay the theoretical ground for designing future machine learning techniques that strive to optimize the discovery capacity in addition to other measures of performance.

Our main results in this section will be Lemma 4.1.5 and Theorem 4.1.6. Lemma 4.1.5 defines the mathematical nature of the *Seen* item space and hence provides tools to explore and control the recommendation strategies defined in that space. Theorem 4.1.6 provides a convergence argument along with a theoretical bound that gives insight about the convergence speed of the user discovery capacity. We start by laying the groundwork in the following Lemma.

Lemma 4.1.4. *Let f be the generic selection function of a recommender system, S_t^c be the complement of S_t, and $g_j \in G$ be a group of items. if f satisfies* ***Assumption 4.1.3*** *and $|S_t| > |Rec_t|$ then*

$$E(|\{g_j \in S_t^c \cap Rec_t\}|) = 0 \tag{4.3}$$

Proof. We first calculate the expected value of the number of recommended item groups. Since we have a fixed set of recommendations, we have:

$$E(|\{g_j \in G \cap Rec_t\}|) = |Rec|,$$

where $|Rec| = |Rec_t|$ $\forall t$. Then

$$E(|\{g_j \in (S_t \cup S_t^c) \cap Rec_t\}|) = |Rec|$$

$$E(|\{g_j \in (S_t \cap Rec_t) \cup (S_t^c \cap Rec_t)\}|) = |Rec|$$

By the linearity of expectation, we have:

$$E(|\{g_j \in (S_t \cap Rec_t)\}|) + E(|\{g_j \in S_t^c \cap Rec_t)\}|) = |Rec|$$

Next, we will find an expression for $P(g_j \in Rec_t)$.

Let us denote the random variable $O = P_f(g_j \in Rel)$ that represents the score (for example predicted normalized rating) of each item group as provided by f, then we can

define a ranking function $h_P : G \to \{0, 1..|G|\}$ that maps g_j to its rank among all groups of items.

Considering the ordered statistics $\{O_{(1)}, O_{(2)}, ..., O_{(|G|)}\}$, we get $h_P(g_j) = q$, where $O_{(q)} = P(g_j \in Rel)$.

Hence, $P(g_j \in Rec_t) = 1$ if $h(g_j) \leq |Rec_t|$ and $P(g_j \in Rec_t) = 0$ otherwise; meaning that an item group is in the recommendation list if its rank is less than $|Rec_t|$. This is based on a purely exploitation strategy that only considers the rank of the relevance estimate of the item.

Therefore, we can affirm that $P(g_j \in Rec_t) = \mathbb{1}_{(h(g_j) \leq |Rec_t|)}$. Thus

$$E(|\{g_j \in S_t \cap Rec_t)\}|) = \sum_{j=0}^{|S_t|} P(g_j \in Rec_t | g_j \in S_t)$$

$$E(|\{g_j \in S_t^c \cap Rec_t)\}|) = \sum_{i=0}^{|S_t^c|} P(g_i \in Rec_t | g_i \in S_t^c)$$

$$|Rec_t| = \sum_{j=0}^{|S_t|} \mathbb{1}_{(h(g_j|g_j \in S) \leq |Rec_t|)} + \sum_{i=0}^{|S_t^c|} \mathbb{1}_{(h(g_i|g_i \in S_t^c) \leq |Rec_t|)}$$

and according to Assumption 4.1.3, we have

$$h(g_j|g_j \in S_t) \leq h(g_j|g_j \in S_t^c).$$

Thus, we have

$$max(h(g_j|g_j \in S_t)) \leq min(h(g_j|g_j \in S_t^c)) \, \forall g_j \in G$$

Assuming that $|Rec_t| \leq |S_t|$, we get $max(h(g_j|g_j \in S_t)) > |Rec_t|$, then $\mathbb{1}_{(h(g_i|g_i \in S_t^c) \leq |Rec_t|)} = 0 \, \forall g_i \in S^c$.

Finally, we conclude that

$$\sum_{j=0}^{|S_t|} \mathbb{1}_{(h(g_j|g_j \in S) \leq |Rec_t|)} = |Rec_t|.$$

This means that

$$E(|\{g_j \in (S_t \cap Rec_t)\}|) = |Rec_t|,$$

and finally

$$E(|\{g_j \in S_t^c \cap Rec_t)\}|) = 0.$$

□

What this lemma is suggesting is that a ranking algorithm satisfying **Assumption 4.1.3** will find an element to recommend from a group that has been seen in order to increase the relevance of the recommendations. We stress again that we are dealing with *groups* of items. This means that the user may see a new item but this item will be from an already seen group. Also note the important role of the recommendation list length, as a long recommendation list may allow for further discovery; but in practice, recommendation lists have a limited length.

This lemma lays the groundwork for the next result, as it allows us to quantify the nature of the Seen Groups Space.

Lemma 4.1.5. *By defining the measurable space* (Ω, G)*, where* Ω *is the sample space of item groups, and if* $E(|\{g_j \in S_t^c \cap Rec_t\}|) = 0$ *; then* S_t *is a filtration on* Ω *and the random process* $|S_t|$ *is a martingale defined in the filtered probability space* $(\Omega, S, \mathbb{P})$*, where* $\mathbb{P}$ *is a probability measure in* (Ω, G)*.*

Proof. the proof that S_t is a filtration is straightforward from the definition since $\forall k < l$, we have $S_k \subset S_l$.

To prove that $|S_t|$ is a martingale, we need to first see that $E(|S_t|) < \infty$ because the number of seen item groups is finite. Then we need to prove that $E(|S_{t+1}||S_t, S_{t-1}, \cdots, S_0) = E(|S_t|)$. First we calculate $E(|S_{t+1}|)$:

$$E(|S_{t+1}|) = E(|S_t \cup Rec_t|). \tag{4.4}$$

$$E(|S_{t+1}|) = E(|S_t|) + |Rec_t| - E(|S_t \cap Rec_t|). \tag{4.5}$$

According to Assumption 4.1.3 ,

$$E(|\{g_j \in S_t^c \cap Rec_t\}|) = 0. \tag{4.6}$$

Therefore

$$E(|S_t \cap Rec_t|) = |Rec_t|. \tag{4.7}$$

Hence

$$E(|S_{t+1}|) = E(|S_t|). \tag{4.8}$$

By the law of total expectation, we can conclude that

$$E(|S_{t+1}||S_t, S_{t-1}...S_0) = E(|S_t|). \tag{4.9}$$

Hence the process $|S_t|$ is a martingale defined in the filtered probability space $(\Omega, S, \mathbb{P})$. □

Theorem 4.1.6. *Let $\Delta S_t = |S_{t+1}| - |S_t|$ be the martingale difference defined on the filtered space $(\Omega, S, \mathbb{P})$. We define the average user discovery $\Delta_n S = \frac{1}{n}\sum_{t=0}^{n} \Delta S_t$. Then, with probability $1-\delta$, we have:*

$$\frac{1}{n}\sum_{t=0}^{n} \Delta S_t \leq \frac{\ln(\frac{1}{\delta})|Rec|^2}{2n}. \tag{4.10}$$

Proof. Since $|S_t|$ is a martingale, it is given that ΔS_t is a martingale difference. Furthermore, we can verify that ΔS_t is bounded. In fact, $\Delta S_t \geq 0$ because the increase is always positive as iterations advance, and also $\Delta S_t \leq |Rec_t|$ since the maximum number of new item groups that are going to be added to $|S_t|$ is the number of recommended item groups. Therefore, using the Azuma-Hoeffding inequality we obtain that $\forall \mu > 0$

$$P(\frac{1}{n}\sum_{t=0}^{n} \Delta S_t > \mu) \leq \exp(-\frac{2n^2\mu}{\sum_{t=0}^{n}|Rec_t|^2}), \tag{4.11}$$

and as the number of groups in the recommended list is constant,

$$P(\frac{1}{n}\sum_{t=0}^{n} \Delta S_t > \mu) \leq \exp(-\frac{2n\mu}{|Rec|^2}), \tag{4.12}$$

If we consider $\delta = P(\frac{1}{n}\sum_{t=0}^{n} \Delta S_t > \mu)$, then solving for μ will give the desired result. □

Lemma 4.1.5 and Theorem 4.1.6 show that the average discovery of the user is decreasing with the number of recommendation iterations n. It also relates this decrease to the length of the recommendation list. This is expected because longer recommendation lists increase the chance to see new groups of items.

We conclude our theoretical analysis with two corollaries that provide convergence results for the average *user discovery* and for the *blind spot*.

Corollary 4.1.6.1. *The average user discovery $\Delta_n S$ converges to 0 almost surely with increasing iterations.*

Proof. The proof is straightforward from the result of Theorem 4.1.6. In fact, using the Azuma-Hoeffding bound, we can show that when $n \to \infty$, $P(\frac{1}{n}\sum_{t=0}^{n} \Delta S_t > \mu) = 0$, and hence $P(\frac{1}{n}\sum_{t=0}^{n} \Delta S_t = 0) = 1$, which concludes the result. □

Corollary 4.1.6.1 further proves that the generic recommender system is rapidly forming a filter bubble *by keeping a blind spot of item groups that is out of reach of the user discovery*. We try to formulate this relationship with Corollary 4.1.6.2.

Corollary 4.1.6.2. *Let $|\Delta B_t| = ||B_{t+1}| - |B_t||$ be the evolution of the blind spot for a given user. We define the average decrease of the blind spot as $\Delta_n B = \frac{1}{n}\sum_{t=0}^{n} \Delta B_t$. Given a recommender system with a decreasing error function $e(t)$ (i.e the accuracy of the recommender system increases after each feedback loop iteration), where $e(t) = |S_t \cap Rel^c|$, then:*

If $\Delta_n S$ converges to 0 almost surely when n tends to infinity, then $\Delta_n B$ converges to 0 almost surely.

Proof. We have:

$$\Delta B_t = \left| \left| S_{t+1}^c \cap Rel \right| - \left| S_t^c \cap Rel \right| \right|. \tag{4.13}$$

Since

$$\left| S_{t+1}^c \cap Rel \right| = |S_{t+1}^c| + |Rel| - |S_{t+1}^c \cup Rel| \tag{4.14}$$

and

$$|S_{t+1}^c| = |G| - |S_{t+1}|, \tag{4.15}$$

we obtain

$$\left|S_{t+1}^{c} \cap Rel\right| = |G| - |S_{t+1}| + |Rel| - |S_{t+1}^{c} \cup Rel|. \tag{4.16}$$

By inserting (4.16) in (4.13) for t and $t+1$, we obtain

$$\Delta B_t = \left|-\Delta S_t - \left|S_{t+1}^{c} \cup Rel\right| + |S_t^{c} \cup Rel|\right|. \tag{4.17}$$

Hence and considering that $\left|S_{t+1}^{c} \cup Rel\right| = |G| - |S_{t+1} \cap Rel^{c}|$, we obtain

$$\Delta B_t = |-\Delta S_t + |S_{t+1} \cap Rel^{c}| - |S_t \cap Rel^{c}|| \tag{4.18}$$

$$\Delta B_t = \left|-\Delta S_t + e(t+1) - e(t)\right|. \tag{4.19}$$

Therefore using the triangle inequality, we deduce that

$$\frac{1}{n}\sum_{t=0}^{n} \Delta B_t \leqslant \Delta_n S + \frac{1}{n}\sum_{t=0}^{n} \left|e(t+1) - e(t)\right|. \tag{4.20}$$

Hence

$$\frac{1}{n}\sum_{t=0}^{n} \Delta B_t \leqslant \Delta_n S + \frac{1}{n}(e(0) - e(n)). \tag{4.21}$$

Since the error is a decreasing function of n, $\Delta_n B$ is positive, and $\Delta_n S$ converges to 0 almost surely when n tends to infinity, $\Delta_n B$ converges to 0 almost surely when n tends to infinity. □

Our theoretical results prove that a collaborative filtering recommender system that aims at increasing its accuracy by recommending relevant items, is prone to forming a blind spot around a certain group of items (depending on the initial state of the system). This result also shows that the existence of the blind spot is dependent on the strategy of the recommender system. In particular, any strategy aiming at only improving the relevance of the recommended items will fail at exploring new items.

4.1.4 Effect of personalization on the human discovery

We discuss the effect of personalization on the human discovery by further studying the effect of Assumption 4.1.3. In fact, Assumption 4.1.3 states that an exploitation focused algorithm will degrade the human discovery and affects their blind spot. Now let us assume

that the recommendation algorithm is a pure random recommendation strategy. Assuming that each group has the same number of items, therefore we would get:

$$P_f(g_j \in Rec) = \frac{1}{G} \forall g_j \in G \tag{4.22}$$

The problem can be formulated as follows: Given the space of all subsets of B_t called $\mathbb{L}_{\mathbb{B}}$. Assuming we are picking an element Z at random from $\mathbb{L}_{\mathbb{B}}$ that has a cardinality of at most $|Rec|$. We can notice that $|Z| = \Delta B_t$. In fact, Z represents the elements of the blind spot that are going to be added to the recommendation list and hence the seen items. Therefore $|Z|$ represents the reduction of the blind spot at the iteration t. We can conclude that $Z = \Delta B_t$.

Proposition 4.1.7. *Given that the recommendation algorithm f is a random strategy. We consider $b_t = |B_t|$ and $k = |Rec|$ Then*

$$E(|\Delta B_t|) = \frac{\sum_{i=0}^{k} i\binom{b}{i}}{\sum_{j=0}^{k}\binom{b}{j}} \tag{4.23}$$

Proof. The result of the proposition is straightforward from the formula of the expectation of a discrete random variable. In fact, given that x is a discrete random variable, we have $E(x) = \sum_i x_i p(x_i)$. In our case, $x = \Delta B_t$ and $x_i = i$ where $0 \leq i \leq k$

Therefore:

$$E(|\Delta B_t|) = \sum_{i}^{k} ip(|\Delta B_t| = i)$$

To evaluate $p(|\Delta B_t| = i)$ we need to notice that there are a total of $\binom{i}{b}$ subset that have a cardinality i. Therefore:

$$p(|\Delta B_t| = i) = \frac{\binom{b}{i}}{\sum_{j=0}^{k}\binom{b}{j}}$$

Finally we get our result:

$$E(|\Delta B_t|) = \frac{\sum_{i=0}^{k} i\binom{b}{i}}{\sum_{j=0}^{k}\binom{b}{j}}$$

□

We can notice that if $b == k$ then $E(|\Delta B_t|) = \frac{b}{2}$. Therefore if the recommendation list is big enough to cover all blind sport groups, the blind spot is expected to be halved at the next iteration.

4.2 Validation of the Proposed Theoretical Model for Recommender System Feedback Loops

4.2.1 Ranking Assumption Validation

We start by empirically evaluating the validity of the Ranking **Assumption 4.1.3** by testing the following null hypothesis, where we use the same notation as in **Assumption 4.1.3**:

Null Hypothesis (H_0): If two item groups have different distances to the seen group set S_t (one group is already seen and the other group is not seen), then their ranking probability P_f is not significantly different.

In order to test this hypothesis, we sample a number of users that have not rated certain groups of items and evaluate the ranking behavior of the algorithm toward these groups compared to the seen groups by comparing the averages of the predicted ratings between the two groups. Finally, we test the hypothesis by performing a t-test to see if there is a significant difference.

In our experiments, we used the Movielens 1M Dataset [1], which has 6040 users, 3952 items, and one million ratings in total. One reasonable way to group items is based on the genres provided with the dataset. There is a total of 18 genres, with several items having more than one genre; thus we verify that the user has already seen **any** of the item's genres. The reason behind this grouping choice is to prevent the recommendation algorithm from explicitly using the item groups in the training. To clarify this further, if we grouped the items by their latent representation, as learned by Matrix Factorization [8] for instance, our test would be biased due to the fact that this segmentation is used in the prediction step. For Matrix Factorization, a five-fold cross validation hyperparameter tuning resulted in the

[1] https://grouplens.org/datasets/movielens/1m/

TABLE 4.2

Comparison between the mean of the predicted ratings for items from the seen groups (same genre) and items from unseen groups.

Item's group	Mean of predicted rating	Variance	P-value
Seen Group	**3.13**	1.5×10^{-5}	2.32×10^{-10}
Unseen Group	3	7×10^{-4}	

following hyperparamters: {Learning rate: 0.001, Latent dimension: 10, L_2 regularization coefficient: 0.01, and number of epochs: 300}. We used Stochastic Gradient Descent (SGD) to train our model and computed the final rating predictions based on the dot product of the user and item latent factor vectors. We further repeated our experiments 10 times, which allows us to compute confidence intervals and perform statistical tests.

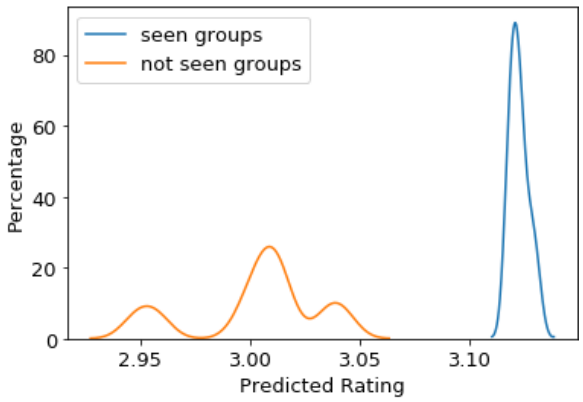


Figure 4.2. **The predicted rating distribution of the seen item groups is significantly different from that of the unseen item groups. The predicted ratings affect the ranking of the items in the recommendation list as stated in Assumption 4.1.3**

The results, shown in **Table 4.2**, lead us to reject the null hypothesis since there is a significant difference (p-value < 0.01) between the two averages. **Figure 4.2** further confirms this difference between the two distributions: The seen item groups have a higher predicted rating on average than the items from the unseen group. This confirms, empirically, that our ranking assumption is reasonable.

What makes this an interesting result is the fact that *we did not use* the genres in the learning and the recommendation step. In fact, the collaborative filtering algorithm was able to detect this dependency based on the similarity of the items' ratings. This is expected as we believe that the recommendation process used to collect the data may have exploited this similarity, and thus exacerbated the limited exposure of the users to different genres.

4.2.2 Empirical Validation of the Asymptotic Behavior of the User Discovery

4.2.2.1 Experimental Design

In order to simulate an iterative recommendation process, we need access to a complete rating matrix. In fact, simulating the sequential dependency needs the true rating of the user to be added at iteration $t-1$ in order to train f_t. Unless we perform real world experiments using a production environment, this is impossible to achieve using the available data due to the high sparsity of the data.

For this reason, we use a semi-synthetic data using the Movielens 1M dataset, as was done in [38, 82]. The process to construct the dataset is done by running a matrix completion algorithm on the original dataset to generate a complete rating matrix. Then, we scale the ratings of each user using a mapping function $g : \hat{R} \rightarrow \{1, 2, 3, 4, 5\}$. By defining five percentiles $(p_1, p_2, p_3, p_4, p_5)$ from the predicted ratings of the user, we apply the following mapping: $g(p_1) = 1$, $g(p_2) = 2$, $g(p_3) = 3$, $g(p_4) = 4$, $g(p_5) = 5$. We remove by this procedure the user specific bias where some users would have a tendency to provide higher ratings or lower ratings compared to others and we scale the predictions into a range that is similar to the original input data. This procedure is similar to what is explained in Section 5.1.2.6

In our simulations, we first assume that the user rates all the recommended items according to the Perfect Feedback **Assumption 4.1.2**. Later, we will explore a relaxation of this assumption where the user sees all the recommended items, but rates a few items based on their ranking in the list, which is similar to [113]. We perform 10 runs and set the

length of the recommendation list to $|Rec_t| = 10$.

4.2.2.2 Results and Discussion

We next investigate the evolution of $|S_t|$ throughout the iterations. **Figure 4(b)** shows how the cardinality of the set of seen item groups stagnates with a negligible increase. Note here that we did not start from the initial state of the system since the available data is collected by running several iterations of recommendations.

We also study the evolution of average user discovery $\Delta_n S$, as shown in **Figure 3(b)**, which confirms the hyperbolic decrease rate in Theorem 4.1.6. This shows that the average user discovery is shrinking throughout the iterations by converging to zero. The variance present in some of the collected points is due to the fact that our data is semi-synthetic and this affects the feedback loop. Despite this sporadically high variance, we can see a general decreasing behavior in $\Delta_n S$.

To further validate our theoretical results, we plot the theoretical bound for two different confidence thresholds. **Figure 3(c)** shows the asymptotic relationship, with increasing iterations, between the empirical result and the theoretical bound.

With these results, we showed that despite not including the groups in the learning process of the algorithm, the recommender system fails to recommend new genres to the user and this is exacerbated when the process advances through more iterations. This decrease explains known phenomena such as the filter bubble or echo chambers [12]. These consequences tend to get worse as a result of the feedback loop.

To investigate the impact of the Perfect Feedback Assumption (**Assumption 4.1.2**), we repeated the experiments by making the user rate only a part of the recommendation list according to a discovery probability [113] related to an item's rank in the recommendation list (the lower the item is in the list, the smaller the chance it will be rated). The results in **Figures 4(c)** show that there is no significant difference, and the behavior of $|S_t|$ and $\Delta_n S$ remains similar to the previous experiment that was depicted in **Figure 4.3**. This shows that even without rating all the items and seeing more items than reporting to the

algorithm, the algorithm is still failing to discover new groups.

In our experiments, we considered a finite number of groups of items which intuitively should discourage the algorithm from creating a filter bubble (If the number is small, then there is a higher chance for the algorithm to discover all items). In fact, in practice, the number of items, in particular taking into account new items that get added continuously, may be huge and thus the cardinality of the item space, when considered over time, can be considered to approach the infinite case. Yet, in our experimental process, we have tested on a finite number of groups. The total number of groups in the Movielens 1M data was below 20. Even in this limited scenario, we could see, as shown in our experimental results, that the algorithm is converging to a state that fails to explore all the groups, again despite the low (finite) number.

4.2.3 Effect of a greedy exploration approach on human discovery

To further challenge our assumptions, mainly **Assumption 4.1.3**, where we assume a pure exploitation based ranking strategy, we limit the validity of the assumption by using a simple exploration strategy. The main objective is to empirically evaluate the impact of **Assumption 4.1.3** on the theoretical findings.

Hence, in this last part of our experiments, we apply a greedy Multi-Armed Bandits (MAB) recommendation strategy [114] and provide observations on the behavior of human discovery. We also evaluate the effect of the degree of the exploration by varying ϵ in the $\epsilon-$greedy MAB approach. Basically, we select a proportion of items in the recommendation list measured by $\epsilon < 1$, where we perform a random selection instead of a ranking approach. This approach violates **Assumption 4.1.3** and hence we cannot predict the asymptotic behavior of the human discovery.

The results in **Figure 4.5** shows that Multi-Armed Bandits strategies are effective in reducing the blind spot of the user. However, the average human discovery increase $\Delta_n S$ does not exceed the theoretical bound. This leads us to think that the naive exploration is not enough to provide the user with a discovery potential that is consistent through

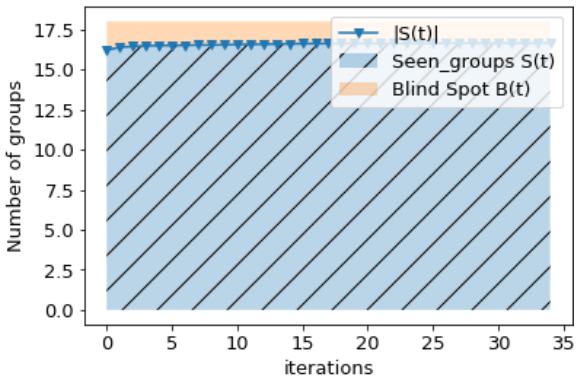
the iterations. Furthermore, this motivates the search for better exploration methods that target the human discovery explicitly by taking into consideration the theoretical behavior of such phenomena.

These findings, although not surprising, as the bias resulting from recommender systems is becoming well known, should lay a theoretical foundation to more targeted methods to solve this problem. Such solutions may for instance, include advanced loss functions that incorporate ΔS into the optimization process, new exploration methods to account for the blind spots, or new algorithms that include the user discovery in their modeling.

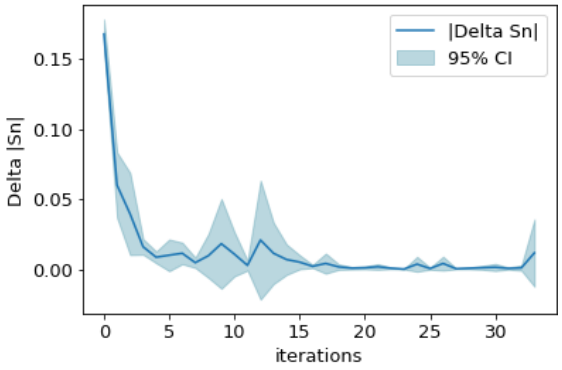
4.3 Summary

In this chapter, we first presented a new theoretical model and study of the feedback loop bias in a collaborative filtering recommender system. We studied the evolution of the user discovery by formally defining the convergence speed and by deriving a lower bound for it. We found that the user discovery can be bounded, in probability, by an exponentially decreasing function. This finding helped us derive a bound for the blind spot of the user as well. We showed that after a few iterations, the blind spot decrease slows down and the blind spot stagnates in a static level.

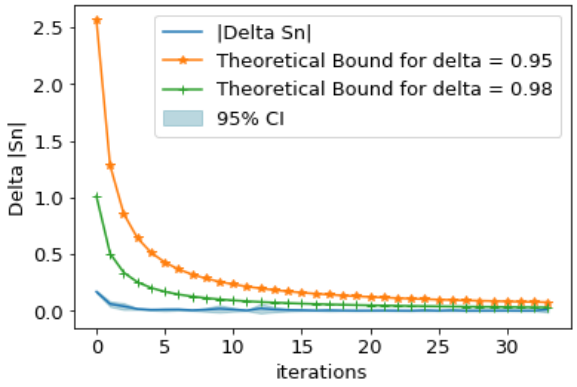
We then evaluated the validity of our new theoretical results by testing the evolution of the user discovery with our assumptions. We repeated the evaluation after relaxing those assumptions. We notice that the recommender systems will converge to a filter bubble after only few iterations. Relaxing the assumption showed that even with exploration strategies, the recommender systems still finds the same static state after few iterations



(a) Evolution of S_t and B_t under perfect feedback

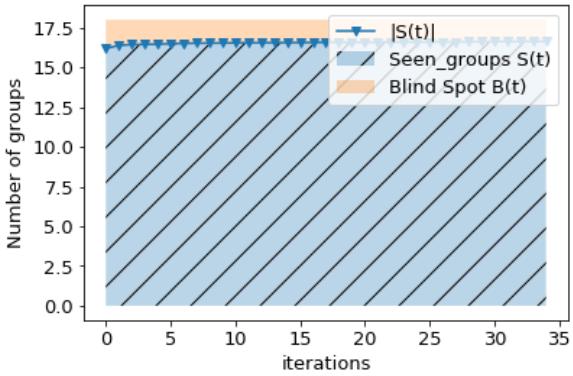


(b) Evolution of $\Delta_n S$under perfect feedback

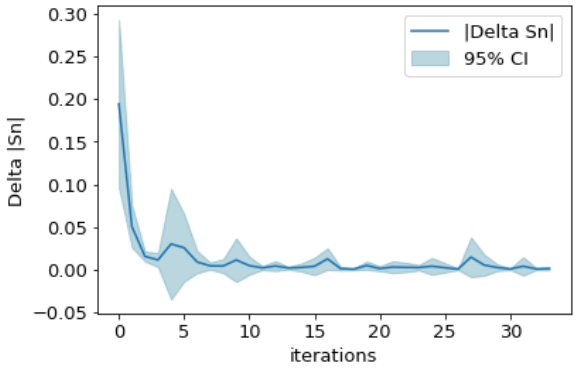


(c) Comparison of $\Delta_n S$ to the the theoretical bound (Theorem 4.1.6) for different levels of δ

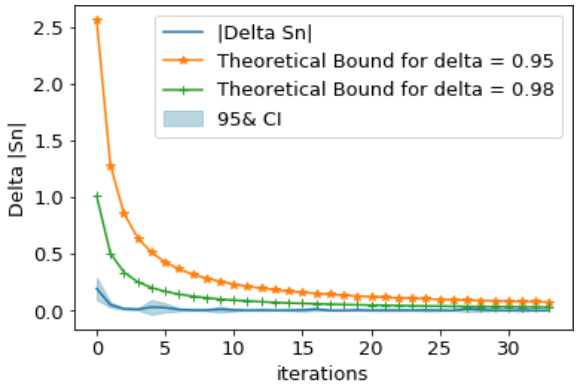
Figure 4.3. **Results with the Perfect Feedback Assumption 4.1.2: (a) shows the evolution of $|S_t|$ through iterations. The shaded area shows the cardinality of the seen item groups (genres) and the non-shaded area shows the blind spot's cardinality. (b) shows the evolution $\Delta_n S$ and how it compares to the theoretical bound with different levels of confidence. (b) clearly confirms the hyperbolic decrease of $\Delta_n S$ in Theorem 4.1.6, with the shaded area representing the 95% confidence interval based on 10 runs.**



(a) Evolution of S_t and B_t under imperfect feedback

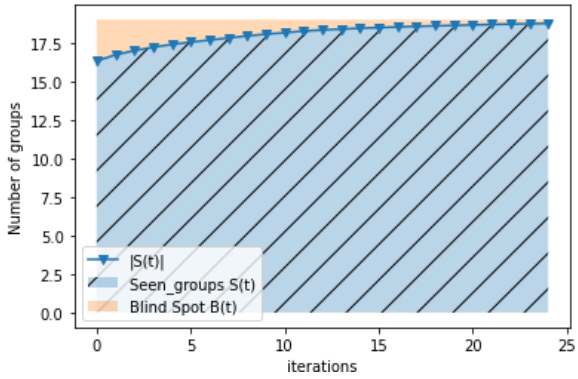


(b) Evolution of $\Delta_n S$ under imperfect feedback

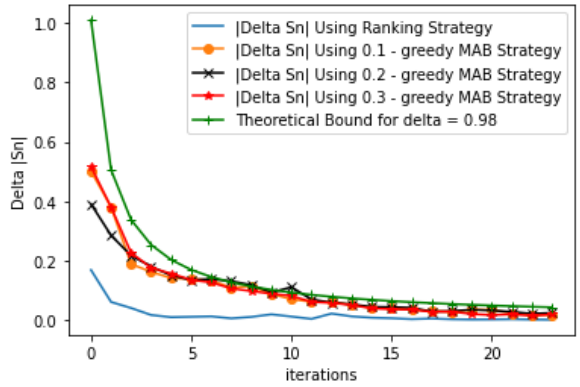


(c) Comparison of $\Delta_n S$ to the the theoretical bound (Theorem 4.1.6) for different levels of δ

Figure 4.4. **Results with the Relaxation of Perfect Feedback Assumption 4.1.2: The user sees the entire recommendation list but only rates a few of them based on a probability that depends on the ranking of the item. We notice that even without the assumption of perfect feedback from the user, the behavior of the algorithm is similar to the previous experience depicted in Figure 4.3**



(a) Evolution of S_t and B_t using ϵ-greedy MAB strategy



(b) Comparison of $\Delta_n S$ to the theoretical bound using $\epsilon-$greedy MAB strategies for different level of ϵ

Figure 4.5. **Results using a greedy exploration strategy with Multi-Armed Bandits. 5(a) shows that using a simple random search will decrease the blind spot size, however the increase rate in the human discovery does not exceed the theoretical bound as shown in 5(b)**

CHAPTER 5

SIMBA: A MODULAR SIMULATION FRAMEWORK FOR STUDYING BIAS IN RECOMMENDER SYSTEMS

In this chapter, we present a new simulation framework aimed at evaluating the **dynamic evolution of bias in recommender systems**.

In Section 5.1 we start by defining the problem of dynamic evaluation in an offline setting. Then we present a deep dive into the design of SimBa by explaining the different modules involved and providing a detailed example on a use case of the simulation. In Section 5.2, we present experimental results showcasing a practical use of SimBa framework in order to evaluate dynamic bias of different common debiasing strategies. We present several research questions and empirically test each one of them.

5.1 SimBa: Design and Methodology

5.1.1 Problem statement

As online testing is expensive and requires access to a large number of users, simulation frameworks represent a good alternative to evaluate recommender systems (RS) in an offline setting. This also allows for more test runs and evaluation iterations. Since RS are complex systems and they involve several moving parts, developing a reliable simulation framework is a trying task. In this chapter, we propose SimBa, a modular decomposition of the recommender system ecosystem, which will facilitate simulations aiming to study bias. This decomposition will allow an easier conception of a simulation framework that can capture the continuous interaction between all the RS components.

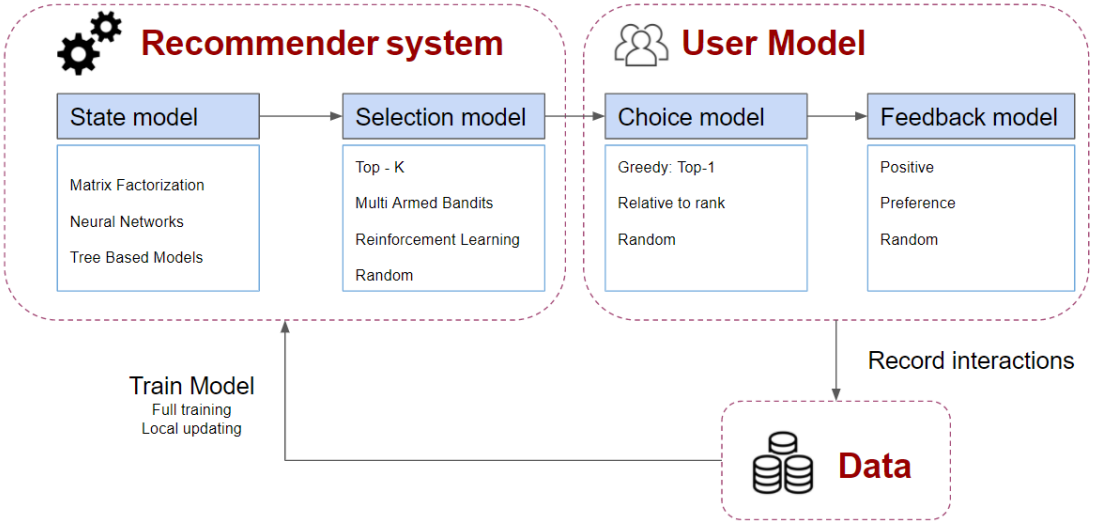


Figure 5.1. Modular decomposition of the proposed simulation framework, into three main modules: Recommender system, User Model, and Data modules. Some modules are in turn decomposed into finer sub-modules. Each module or sub-module can be manipulated independently of the remaining components

5.1.2 Proposed design

We define the *recommender system ecosystem* as the combined interactions of three main modules, as shown in Figure 5.2: (1) The recommender system (algorithm), (2) the user, and (3) the data. The following sections describe each module.

5.1.2.1 Recommender system module

The recommender system module is itself decomposed into two sub-modules, namely a state model and a selection model, as described below.

5.1.2.2 State Model

Definition 5.1.1. *Without loss of generality, the RS state model can be composed of (1) a mapping function* $M : U \times I \rightarrow R^d$ *that maps the users* U *and items* I *to a shared embedding space* R^d *with a given dimension* d*; and (2) a scoring function* $f : R^d \rightarrow R$ *that is used to score the user-item interactions (such as by using the dot product or a Multi-Layer*

Perceptron).

Most recommender systems model the users and items in a shared latent space through learning embeddings. These embeddings represent the state of the recommender system. Models to learn these embeddings differ depending on the scale, application, and data available to the algorithm. For instance, Collaborative Filtering algorithms such as Matrix Factorization [8], use the interaction between the users and items to learn embedding vectors by exploiting latent similarities through dot product relation. Other methods, like NeuMF [115] and other deep learning-based models, uses deep neural networks to learn embeddings of the users and items. The deep learning architecture varies depending on the application and the targeted performance (Transformer based models [51], CNN, LSTM [116], etc). Content-based Filtering models (CBF) rely on the item and user features in order to predict the likelihood of the interaction [21]. These methods often rely on raw and engineered features in order to represent the state of the system. Deep Learning methods can also be used to map the users and items features to a latent space [116]. Hybrid models combine CF and CBF [117], for instance by combining the user-item interactions with external information for more powerful representations.

In our simulation framework, the state model's role is to learn user and item embeddings that can be later used to generate the recommendations. The state model is a sub-module that is independent of the remaining modules of the framework and can be changed according to the need and the tested hypothesis.

5.1.2.3 Selection model

Definition 5.1.2. *A selection model is a recommendation policy π that maps the calculated scores from the state model to rankings, according to specific strategies such as greedy, MAB, random, etc.*

The next step after learning the state model is to select recommended items. In order to filter a handful of items, several strategies are adopted. For instance, the top-K greedy strategy selects the top K items based on the highest-ranked items using the output

scores from the state model. This method has been criticized for being an exploitation-only strategy that hinders user discovery. Post-processing methods and re-ranking have been used in order to improve the diversity of the items [118].

Other strategies introduce some exploration such as Multi-Armed Bandits (MAB) strategy [63]. MAB optimizes for long-term satisfaction through different exploration techniques.

Another recent trend in selection strategies is using Reinforcement Learning to learn effective long-term satisfaction. Reinforcement Learning strategies use state model outputs (i.e embeddings) in order to represent the environment state and then optimizes a policy π that selects the recommended items [119]. User feedback is used to optimize the policy π. The challenge of RL strategies comes from the difficulty in learning such methods in an offline setting. In fact, in order to simulate proper feedback, online interaction with users needs to be set up. There exist however offline policy learning techniques. This area is an ongoing research field and we do not consider these strategies at this time.

We should also mention that, depending on the scale of the system, the recommender system model can be further decomposed into multiple stages. In fact, most deployed recommender systems rely on a two-stage selection process by employing lighter models in the first stage to select candidate items, followed by another more complex model to select the final recommended items [37]. Without loss of generality, we currently use a single-stage recommender system.

5.1.2.4 User module

The most challenging part of a simulation strategy is modeling the user's behavior. In fact, the recommender system component can be controlled and modeled to mimic the real behavior of a production system according to the available data. On the other hand, the user's behavior is affected by external factors that cannot be predicted. These factors include the user's personal bias which affects the feedback, the user's social circle which affects the exposure, and even the user's mood which creates a dynamic feedback model.

These factors and their interaction are hard to model. Therefore, assumptions should be made to simplify the process while staying truthful to the real-world behavior, as follows.

Assumption 5.1.3. *The user model is composed of (1) a choice model that selects the items to interact with, and (2) a feedback model that provides feedback according to the selected items.*

Assumption 5.1.3, adopted in [103], allows the creation of different models for both choice and feedback. This in turn allows the simulation to act on the user's personal bias, user's exposure, and the different layers of interactions between the user and the recommender system.

5.1.2.5 Choice model

The choice model for the user defines the user interaction with the presented recommended items. The choice model depends on several factors, such as the way the items are displayed (grid, ranked list, a news feed, carousel, etc) and the preference of the user. Choice models include several options such as (1) a deterministic greedy approach where higher scored items are chosen before items with lower predicted score; (2) a random approach where the choice of the user does not depend on the rank and interacted items are sampled randomly from the ranked list; or (3) a probabilistic model where higher-ranked items have a higher probability of being selected.

In our design, we propose a click model inspired from economic theory studying Trial and Offer markets or ***T-O*** (a setting where consumers can try products before deciding to buy them), where the click decision (analogous to trying a product in T-O) is inspired from the social influence, position and quality of the item. We adopt the logit model used in [120] defined as follows:

Assumption 5.1.4. *The probability of the user u clicking an item i is given by:*

$$P(u \text{ clicks } i) = \frac{\hat{r}_{ui} v(\sigma_i)(\Phi_i)^\alpha}{\sum_{j \in I} \hat{r}_{uj} v(\sigma_i)(\Phi_j)^\alpha} \tag{5.1}$$

Where:

- $\hat{r}_{ui}$ is the predicted rating or relevance (analogous to the inherent quality in T-O) of item i to user u.
- $v(\sigma_i)$ is a visibility function of the position of the item, capturing the decreasing attention of the user to an item that appears in lower positions. σ_i defines the position bias.
- Φ_i defines the social influence (analogous to market share in T-O) of item i and is correlated with the popularity of the item.
- α is the strength of social influence (analogous to the social signal in T-O). It represents the nonlinear effect of the social influence of the item on the user u. The non-linearity in this case is modeled by an exponential function, but it can also be modeled by other non-linear functions.

The idea of the proposed choice model is that the items present in the ranked list shown to the user compete for the user's attention. The model takes into account this competition by a logit function and depends on the quality, position, and social influence of the item and other alternatives in the same list.

The choice model allows us to *control* for various types of bias such as position bias and popularity bias. This control mechanism will in turn allow us to test the effect of various models on these biases.

After generating the probability scores for all items **present** in the recommended list, we sample a random item based on these probabilities to determine the item *clicked* by the user. The next step would be to figure out the *feedback* given by the user on this item.

5.1.2.6 Feedback model

After choosing an item, the user typically provides feedback. This feedback can be explicit through ratings or likes and dislikes, or it can be implicit through clicks. The user feedback usually depends on the relevance of the item and the preference of the user.

In our simulation, we adopt semi-synthetic data completion strategy [16, 38, 121]. The idea is to create a mapping of (users, items, ratings) where the ratings are completed by a matrix completion algorithm such as Matrix Factorization [8]. The predicted ratings are then transformed to percentiles and each percentile is mapped to a single rating. Ratings are discrete and range from 1 to 5.

After constructing the semi-synthetic data, we use it to simulate user feedback on the recommended items. This feedback is then added to the original dataset.

5.1.2.7 Data module

The data module handles the input and outputs to and from the other modules. In fact, the recommender system needs to be retrained regularly in order to reflect the previous feedback, and the new ratings need to be added to the original data. The data usually consists of user-item interactions in a typical Collaborative Filtering setting. External data such as user and item features can also be used. Some large-scale recommender systems also store embeddings as data to feed to the recommender system module. In our current simulation, we adopt a Collaborative Filtering approach; therefore the data used is a list of user and item interactions, with the ratings being explicit ratings.

5.1.3 Example: Simulation module for classic Matrix Factorization

In this section, we provide a basic example of how our simulation framework can be used to model a classic recommender system. We use Matrix Factorization as a state model and use a greedy top-k selection method as a recommendation policy.

5.1.3.1 State model: Matrix Factorization

In the following experiments, we will use Matrix Factorization as our state model in order to learn the user and item embeddings. Matrix Factorization learns two vectors P and Q of a latent space dimension d. The items are then scored using a function $f()$, for

instance a dot product:

$$\hat{r} = f(P, Q) = P.Q^T \tag{5.2}$$

The model is learned using a Mean Squared Error loss (MSE) since we are using explicit feedback for the task of rating predictions. Defining O as the set of the user-item ratings,

$$J = \frac{1}{|O|} \sum_{(u,i) \in O} (r - P.Q^T)^2 \tag{5.3}$$

Our simulation framework allows the freedom to define the desired loss function alongside the model implementation. Hence, other losses can also be tested, such as the BPR loss [42] or the binary cross-entropy loss for learning binary interactions. Our simulation framework also allows the choice of a debiasing strategy. For instance, debiasing strategies, that can be applied on the base Matrix Factorization model, include using an IPS-based loss [38] or applying a MAB strategy as a selection model using the predicted ratings.

5.1.3.2 Selection model: Greedy selection

The next step after computing the predicted rating score $\hat{r}$ is to rank the items using a ranking strategy. The greedy approach is an exploitation-based strategy that ranks the items according to their predicted scores and then selects the top k items to present to the user. Other strategies that might be explored here include Multi-Armed Bandits, Reinforcement Learning, or simply a shuffling strategy that shuffles the top-k items.

5.1.3.3 User model: Logit choice model and semi-synthetic feedback

For the user model, we use a logit choice model where the user interacts with all the presented items. Other models might be used such as a cascade model where the user browses the item list one by one until clicking on one item. For the feedback model, we use semi-synthetic feedback as the feedback model. Data is completed using a well tuned Matrix Factorization and then predictions are transformed to synthetic data.

5.1.3.4 Data: Explicit feedback + No external data

We use an explicit feedback setting where the ratings are non-binary. We also use only user-item interactions. We frame our problem in a Collaborative Filtering setting where the model is only allowed to use ratings as training data. The reason behind this choice is the need to isolate the effect of the ratings on the recommender system pipeline. Features that could be derived from the social network of the user, item information, etc, may introduce additional bias into the system.

5.1.3.5 Putting everything together

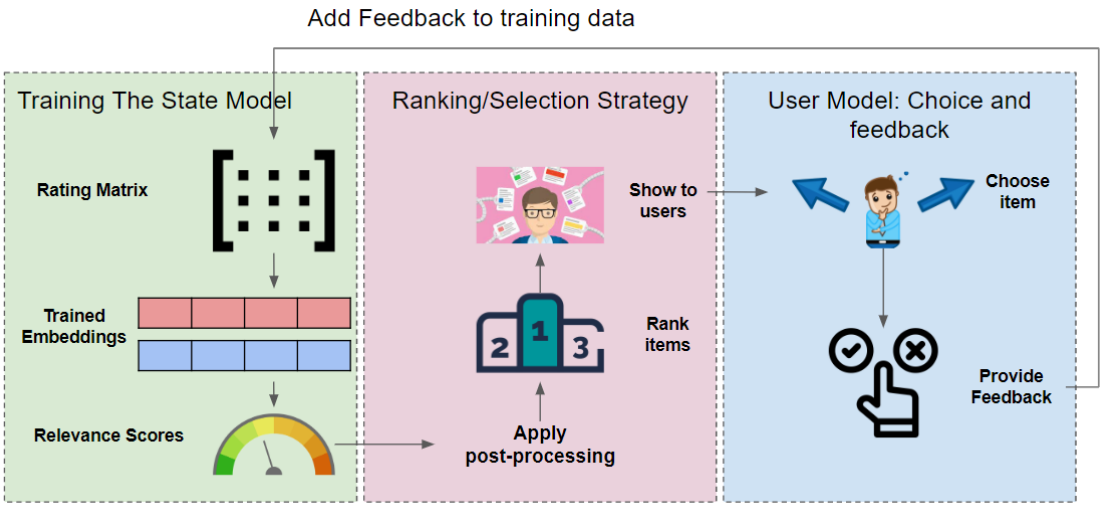


Figure 5.2. The simulation process is divided into three main components corresponding to the different stages. First, we start by training a state model such as Matrix Factorization in order to compute user and item embeddings and items scores. These scores will be used to generate a ranking of the items that will be presented to the user. The next step is to generate a choice model based on a probability distribution. The chosen item will be assigned a feedback score taken from the semi-synthetic data and added to the original dataset.

The full simulation pipeline is described in Algorithm 5.1 and illustrated in Figure 5.2. We start by training the initial embedding for users and items. The hyperparameters are tuned using an independent validation pipeline. After obtaining the training embeddings that represent the state of the system, we calculate the predicted ratings $\hat{r}$ using the dot

Algorithm 5.1 Full Simulation Steps

Input: Ratings R, number of iterations, *iterations*, semi synthetic data $R_{ui}^{semi-synth}$, state model P, Q,
Output: $\tilde{r}$
for $t = 1$ **to** iterations **do**
  Train P, Q // Train Matrix factorization model for iteration t.
  $\hat{r} = f(P, Q)$ // Calculate predicted score
  $Rec_t = \pi(\hat{r}, P, Q)$ // Select top-k recommendations.
  $\Phi_i = \frac{\mathbb{1}_{r_{ui}}}{|U|}$ // Calculate popularity scores
  $P(u \text{ clicks } i) = \frac{\hat{r}_{ui}\sigma_i(\Phi_i)^\alpha}{\sum_j \hat{r}_{uj}\sigma_j(\Phi_j)^\alpha}$ // Calculate the Choice probabilities
  $Selected_t \sim P(u \text{ clicks } i)$ // Select items from the recommendation list by sampling from Choice probabilities.
  $\tilde{r}_{ui} = R_{ui}^{semi-synth}$ // Apply a feedback model to each selected item from the semi synthetic data.
  $R_{t+1} = R_t \cup \tilde{r}$ // Add ratings to the initial training data
end for

product. Then we rank the items according to their scores and select the top-k items to present to the user (in our experiments we use $k = 10$). A click probability is then calculated according to Equation 5.1. These probabilities will be used to sample the chosen item. After sampling the chosen item, the synthetic feedback is added from the semi-synthetic data created.

We retrain the state of the system every iteration. The reason for this choice is that since the user is selecting all k recommended items, at each iteration, a large number of ratings get added to the data and the state of the model should be updated.

5.2 Demonstrating the use of the SimBa simulation framework for feedback loop analysis and dynamic debiasing

In this section, we demonstrate the use of the proposed simulation framework to study the effect of different debiasing strategies in a feedback loop setting. We first study the interaction of three different debiasing strategies, by conducting experiments with different debiasing strategy combinations and using several bias evaluation metrics to assess their effect. We then explore the effect of the initial bias in the data on the evolution of the data. We use an unbiased real-world data set in order to investigate this effect.

5.2.1 Research Questions

The modular structure of our simulation framework allows the isolation of the different components and effective testing of different research questions. In this study we try to answer the following questions:

- **RQ5.1** Does the Inverse Propensity Scoring (IPS) strategy solve popularity bias and diversity bias?
- **RQ5.2** Does the Multi-Armed Bandit strategy (MAB) solve exposure bias?
- **RQ5.3** Does combining IPS and MAB strategies help further reduce the bias of the system?
- **RQ5.4** How do IPS and MAB affect the bias of the system in a feedback loop setting?
- **RQ5.5** How do IPS and MAB affect the accuracy of the system in a feedback loop setting?
- **RQ5.6** Does breaking the feedback loop bias by providing random recommendations help reduce the bias of the system in subsequent iterations?
- **RQ5.7** How does the initial level of bias in the training data affect the answer to RQ5.1 - RQ5.4?

Table 5.1 shows the effectiveness of our simulation framework in order to test the proposed research questions. In fact, thanks to the modularity of our approach, we can use different training strategies such as IPS in order to train the state model with Propensity weighted debiasing [38]. We can also combine the model training with different debiasing and selection strategies, for instance by combining different post-processing-based debiasing methods [122] and selection strategies. Previous simulation strategies [94–98] have focused mainly on the selection strategies using Reinforcement Learning frameworks and neglected the importance of the other crucial components such as the choice model and the state model.

5.2.2 Data

We use two publicly available real-world datasets for our experiments.

- **Movielens 100K[1]:** The Movielens data is a popular benchmark dataset of movie ratings that contains 943 users and 1682 items and 100,000 ratings. The ratings are scaled from 1 to 5.
- **Yahoo! R3[2]:** The Yahoo! R3 Dataset is an unbiased music rating dataset. It contains 15,400 users, 1000 songs, and 54000 unbiased ratings. We use this dataset in order to evaluate the effect of the feedback loop and debiasing strategies when starting from an unbiased setting.

5.2.3 Debiasing strategies

We use different debiasing strategies, described below:

5.2.3.1 Inverse Propensity Scoring (IPS)

IPS-based methods aim at removing the statistical bias caused by the biased exposure of the user. It is used in the training phase by modifying the loss function. For the case of the MSE loss, considering O as the set of observed ratings and J as a loss function defined by:

$$J = \frac{1}{|O|} \sum_{u,i \in O} (r - \hat{r})^2, \tag{5.4}$$

it can be shown that J is a biased estimator of the true loss J_{true} of the system, meaning that $E(J) \neq J_{true}$. The IPS strategy uses propensity scores P_{ui}, that represents the probability that user u is exposed to item i, to down weight the items that have high propensity and eliminate the statistical bias from the loss, as follows

$$J_{IPS} = \frac{1}{|U||I|} \sum_{u,i \in O} \frac{1}{P_{ui}} (r - \hat{r})^2 \tag{5.5}$$

[1] `https://grouplens.org/datasets/movielens/100k/`

[2] `https://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=3`

As propensity score, we employ the popularity score of each item, although a more complex propensity model could also be implemented within our framework in future studies. As IPS is designed to reduce exposure bias, its effect on other types of biases such as diversity or popularity bias is still unexplored. We use our simulation framework to investigate this effect.

5.2.3.2 Multi-Armed Bandits

MAB-based strategies are powerful debiasing techniques that introduce exploration in the recommended items list. The idea behind MAB is to explore a good exploitation-exploration trade-off. In our experiments, we use an $\epsilon-$greedy approach. This approach introduces a uniform noise to the recommendation list. MAB strategies have already shown effectiveness in increasing the diversity of the recommendations, but using our simulation framework, we will also be able to study the effectiveness of MAB on popularity bias and exposure bias. Furthermore, we will study the effect of ensembling MAB strategies with other debiasing techniques.

5.2.3.3 Breaking the feedback loop

Studies have suggested that in order to reduce the bias of recommender systems in a feedback loop setting, we need to break the loop by recommending random items in a given iteration [70]. Using the full exploration approach will certainly allow for diverse and unbiased recommendations during this given iteration, but how does this method affect the behavior of the recommender system in subsequent iterations? We investigate this effect, alongside the interaction with other debiasing strategies, by breaking the feedback loop several times during our simulation.

5.2.4 Bias evaluation metrics

We evaluate the effect of each debiasing strategy on various types of bias using bias evaluation metrics that measure three main aspects of bias: popularity, novelty, and

diversity.

5.2.4.1 Intra-list Diversity (ILD)

This metric [108] allows assessing the diversity of the recommended list by measuring the average pairwise distances (or similarities) between all the items, where the similarity is based on how they are rated. Considering *Rec* as the recommendation list (selected items to be presented to the user) then

$$ILD = \frac{2}{|Rec|(|Rec|-1)} \sum_{i,j \in Rec, i \neq j} 1 - cosine\_similarity(i,j).$$

5.2.4.2 Min Intra-list Diversity (Min-ILD)

As ILD measures the *average* diversity of the list, it can be biased if the recommended items contain outliers such as when only one or two items are different and all the other items are very similar. Min-ILD counteracts this bias by assessing the *minimum* guaranteed diversity in the recommended list. The Minimum Intra-List Diversity [109] calculates the minimum pairwise distance within a recommended list as follows:

$$Min - ILD = \min_{i,j \in Rec, i \neq j} 1 - cosine\_similarity(i,j).$$

5.2.4.3 Average Popularity

In order to evaluate the algorithm's coverage of the long tail items, we use the average popularity metric defined as

$$Avg\_Pop = \frac{1}{|Rec|} \sum_{i \in Rec} Pop_i.$$

This metric allows the estimation of the percentage of the popularity level in a recommendation list and therefore evaluates the effect of the entire recommendation pipeline on the popularity bias.

5.2.5 Measuring Both Relevance and Bias using the Expected Popularity Complement (EPC)

Expected Popularity Complement (EPC) [113] assesses the expected novelty of the recommendations by taking into account their *relevance* as well. In this work, we measure EPC using

$$EPC = \frac{1}{|Rec|} \sum_{i \in Rec} (1 - P_{ui}).\frac{1}{log(rank + 2)},$$

where P is the propensity, considered in this work to be the same as popularity; and the logarithmic term accounts for the rank of the item, analogously to the NDCG metric. We basically want items with higher rank to have a lower propensity.

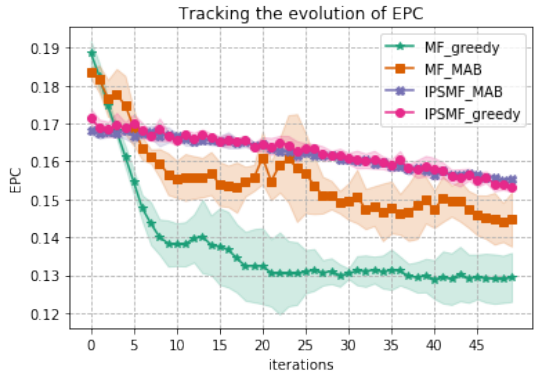
5.2.6 Implementation details

We used Pytorch to implement Matrix Factorization using the IPS loss and perform hyperparameter tuning using Optuna [1] with Tree-structured Parzen Estimator for each dataset. The tuning was only performed at the beginning of the simulation experiments. The hyperparameters for each model are summarized in Table 5.2. For the MAB strategy, we use $\epsilon-$greedy with $\epsilon = 0.1$. This means that for a recommended list of 10 items, we include one random item. Each simulation experiment was run 5 times and the results are reported with their 95% Confidence Interval to assess the significance. The simulation was run for 50 iterations. Each complete run takes around 9 hours on a V100 Nvidia card. No preprocessing was done on the data and we kept the initial level of sparsity and the same initial rating distribution.

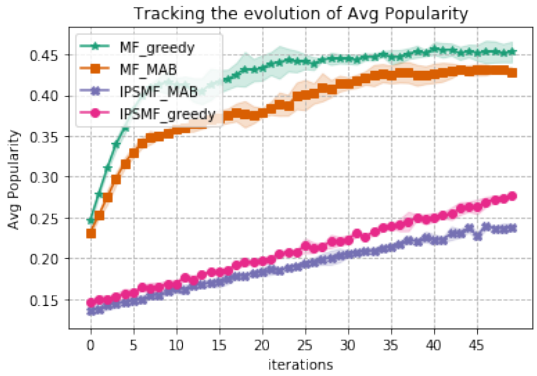
5.2.7 The joint effect of the Debiasing Strategies on the recommender system Feedback loop (RQ5.1 - RQ5.5)

Figure 5.3 shows the evolution of the different bias metrics that are tracked using our simulation framework. **We observe that adding MAB significantly improves the**

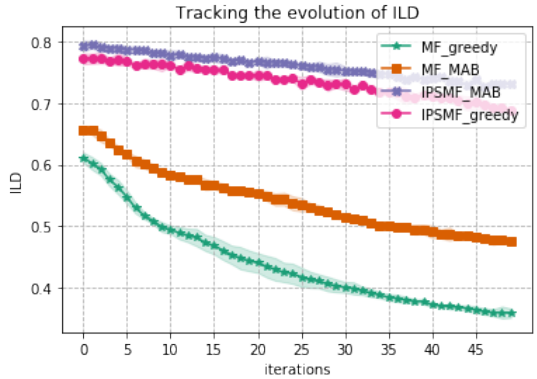
[1] `https://optuna.org/`



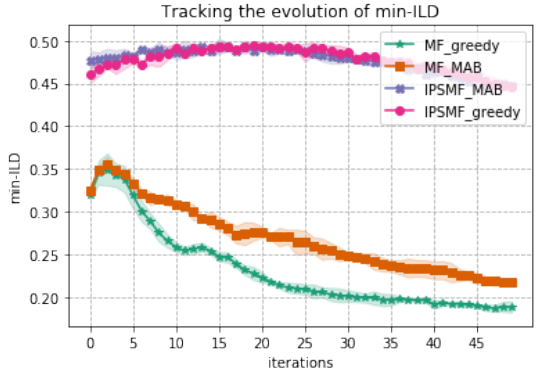
(a) Evolution of EPC for Movielens 100K



(b) Evolution of the Average Popularity for Movielens 100K



(c) Evolution of the Intra List Diversity for Movielens 100K



(d) Evolution of the Minimum Intra List Diversity for Movielens 100K

Figure 5.3. The evolution of the bias evaluation metrics for the Movielens data. We notice that using only IPS for debiasing does not increase the diversity of the system. IPS does however improve the popularity coverage of the system. We also observe that combining IPS and MAB improves the novelty. The shaded areas represent 95% confidence intervals based on 10 runs.

diversity and coverage of the results when using MF as a state model (RQ5.1). **However, combining MAB and IPS do not seem to improve the results (RQ5.3)**. In fact, looking at the ILD and the Min-ILD evolution, we notice that IPS-MF and IPS-MF with MAB have similar evolution. **We also observe that using only IPS for debiasing does improve the diversity and popularity bias (RQ5.1)**. Looking at the EPC evolution, we notice that IPSMF successfully improves the EPC metric, as expected, since the propensity scores are taken into account while learning the embeddings so the recommendations will be more accurate. Another observation is that **adding MAB also improves the novelty of the recommendations (RQ5.2)**. MAB, however, introduces

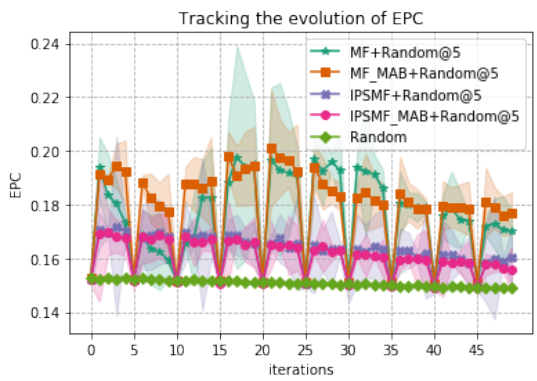
more variance which is expected due to the randomness introduced in the recommendation process. **We also notice that combining MAB and IPSMF does not improve the EPC metric over using IPSMF alone (RQ5.3).**

Next, looking at the iterative evolution of the system, we clearly observe a bias amplification in all the bias metrics. The effect is more clear in the diversity and popularity metrics. The novelty metric incurs a larger drop at the start of the simulation, but stagnates after a few iterations. We also observe that IPS reduces the bias amplification while not totally preventing it. We confirm that, according to our simulation, **the IPS and MAB strategies do not prevent the bias amplification in the feedback loop setting but IPS manages to reduce its effect (RQ5.4).**

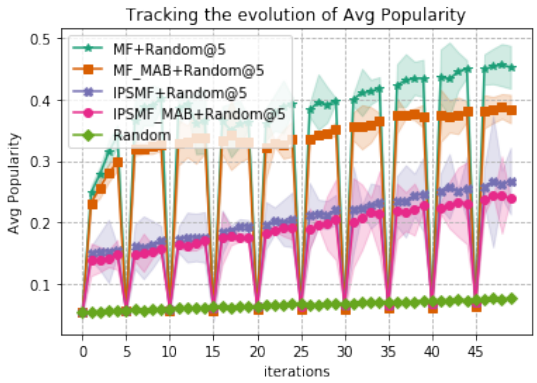
We also notice from the EPC plot in Figure 5.3 that IPS debiasing strategy provides the best accuracy. Furthermore, It is more stable compared to MAB and greedy selection. We also notice that combining IPS and MAB does not affect the accuracy of the system. We should also notice that the accuracy measure in our experiments are subject to the bias of the semi-synthetic data that has been introduced. In order to measure the accuracy of the system in an unbiased manner, we should run live experiments with real user behavior which is out of the scope of this work.

5.2.8 The effect of breaking the feedback loop through random recommendations (RQ5.6)

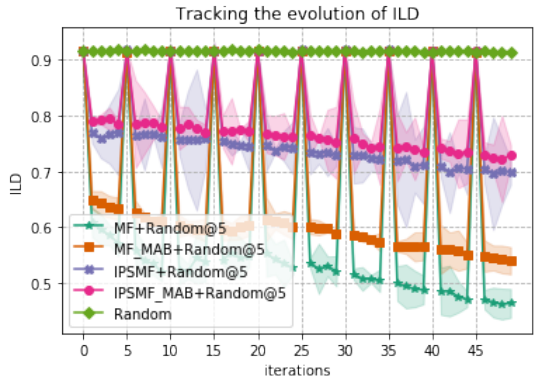
In the following experiments, we simulate breaking the feedback loop periodically by injecting periodic random recommendations, where we provide totally random recommendations every five iterations. The idea is to study the effect of breaking the feedback loop by adopting a full exploration strategy in the middle of the simulation. Figure 5.4 shows the evolution of the system under this strategy. We clearly see an oscillating behavior in all the bias metrics. We also plot a totally random recommendation policy as a baseline. We notice that every five iterations, every metric bounces back to the random strategy level. The difference in performance for the models did not change, IPS still performs better than



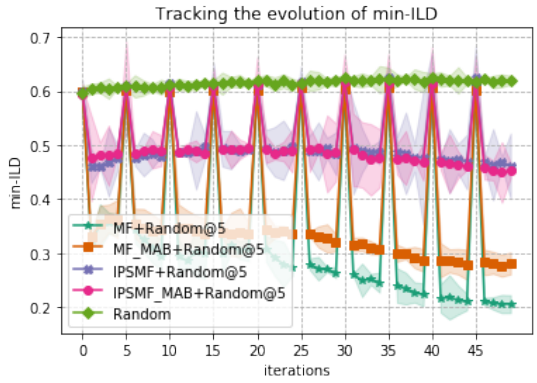
(a) Evolution of EPC for Movielens 100K with random recommendations every 5 iterations



(b) Evolution of the Average Popularity for Movielens 100K with random recommendations every 5 iterations



(c) Evolution of the ILD for Movielens 100K with random recommendations every 5 iterations



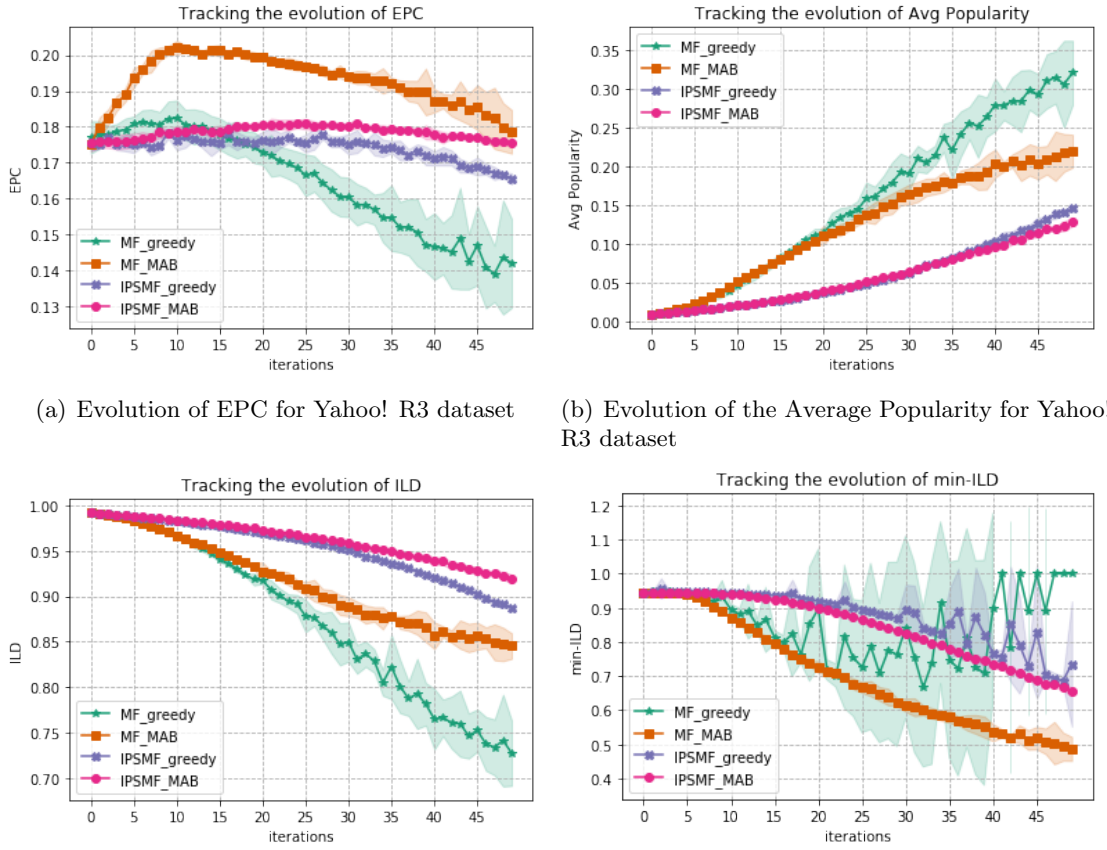
(d) Evolution of Min-ILD for Movielens 100K with random recommendations every 5 iterations

Figure 5.4. The evolution of the bias evaluation metrics for the Movielens data while injecting periodic random recommendations, hence simulating breaking the feedback loop periodically. We notice that breaking the feedback loop does not help reduce the bias of the system and does not solve the iterative bias problem. The figures show in fact that the bias not only persists but still gets amplified. The shaded areas represent 95% confidence intervals based on 10 runs.

the status quo (vanilla) model training, and combining IPS with MAB did not affect the performance. One interesting observation though is the behavior of the EPC metric. In fact, we notice that the model's performance exhibits large variance and all models. This may be due to the inaccuracy of the results introduced by the random iterations and therefore, the variance increases. **We also observe that in the iteration that follows the random recommendations (where the feedback loop is broken), the bias persists and goes back to the original level without much effect on the feedback loop results (RQ5.6) and the amplification of bias persists as well.** Considering that

random recommendations cause a significant drop in the accuracy of the system resulting in reducing user satisfaction and hence reducing the interaction frequency with the system, we conclude, according to our simulation setting, that introducing pure exploration strategies in the feedback loop is not an effective strategy to reduce the feedback loop bias effect.

5.2.9 The effect of the initial dataset bias on the feedback loop (RQ5.7)



(a) Evolution of EPC for Yahoo! R3 dataset

(b) Evolution of the Average Popularity for Yahoo! R3 dataset

(c) Evolution of the Intra-List Diversity for Yahoo! R3 dataset

(d) Evolution of the Minimum Intra-List Diversity for Yahoo! R3 dataset

Figure 5.5. The evolution of the bias metrics for the Yahoo! R3 dataset. We observe that starting from an unbiased setting helps stabilize the system, but does not completely prevent the amplification of the bias due to the feedback loop. The shaded areas represent 95% confidence intervals based on 5 runs.

We use the Yahoo! R3 music dataset to evaluate the effect of the initial bias in the data on the feedback loop. When starting from unbiased data, which is the case in the Yahoo! R3 dataset, we observe the evolution of the evaluation metrics for different

debiasing strategies, as shown in Figure 5.5. We can see that the feedback loop still affects the performance of the algorithms, although the evolution of the system differs from the one observed with the Movielens data. To start, we notice that the early five iterations result in similar performance across all the methods in contrast to when we used the Movielens data in Figure 5.3. This is due to the presence of unbiased data (unaffected by a recommendation strategy) while training. Therefore, the model will learn a true representation in the early iterations. However, as iterations increase, we notice that methods diverge from each other and behavior differ. Matrix Factorization using a greedy strategy, as expected, performs the worst. We see a significant bias amplification with high variance behavior especially with the diversity metrics.

We also notice that MAB strategy improves the EPC metric but fails to improve the populairty and diversity of the model and behaves similar to MF. IPS-MF is the most stable model among other debiasing strategies. Although it fails to improve the bias or prevent the feedback loop effect as iterations advance, it performs the best in terms of stability. We notice that the popularity of the system increases at slower rate than other methods. We also notice the same behavior as in Movielens data where combining IPS and MAB do not have a significant effect in early iterations but it does seem to improve results in latter iterations.

We conclude from these experiments that the initial bias in the dataset does affect the bias evolution in the feedback loop. In fact, an unbiased initial setting does not help with debiasing the feedback loop in the long term but does stabilize the system in terms of bias in the early iterations.

Most importantly, we have demonstrated how the proposed SimBa simulation framework provides a flexible and modular framework to explore a large (in fact, combinatorial in the number of choices) variety of Research Questions, of which we have only shown a small sample.

5.2.10 Limitations and future work

We proposed a modular recommender system simulation design that incorporates state model training as well as user interaction modeling, we then illustrated how to use the simulation framework on two real-world datasets, in order to study the feedback loop under different settings. We explored the effect of mixing different debiasing strategies, the effect of these debiasing strategies on the feedback loop, and the effect of the initial level of bias in the data on the feedback loop.

One limitation of our work, in its current stage, is not using more complex user modeling. In fact, in order to reduce the number of possibilities, as the number of parameters grows exponentially, we only experimented with common settings that are used in the literature. Our work can be extended to include more user models, more debiasing strategies, and even a Reinforcement Learning setting.

The need to audit RS algorithms in a safe and effective way, in terms of the bias effect and the feedback loop evolution is crucial. Our work can be used in order to test various recommender system designs in terms of bias, and potentially measure further social impacts if these can also be modeled. Another future use of our simulation framework is to test for butterfly effects of simple changes in the RS pipelines such as data preprocessing, hyperparameter settings, or user behavior. Another important question that can be asked is whether complex behavior arises from incremental changes, and how to effectively test for these effects?

5.3 Summary

In this chapter, we proposed an evaluation framework, SimBa, that consists of a modular simulation of the recommender system ecosystem. We detailed each component, mainly the state model training, the selection strategy and the user modeling including the choice model and feedback model.

This framework can be used to assess the effect of recommender system models on bias. It also can track the bias of the system in a well controlled environment, which allows

flexibility in testing various model configurations, as well as selection and choice models, in a modular way.

We showcased a use case for our simulation framework by evaluating common debiasing strategies in different empirical settings and test different research questions. We provided an extensive analysis of the performance of these debiasing strategies on two different datasets. Most importantly, we have demonstrated how the proposed SimBa simulation framework can offer a flexible toolkit to explore a large (in fact, combinatorial in the number of choices in each module) variety of Research Questions.

TABLE 5.1

Effectiveness of our simulation framework at testing research hypothesis compared to other available simulation frameworks

Simulator	RQ5.1	RQ5.2	RQ5.3	RQ5.4	RQ5.5	RQ5.6	RQ5.7
RecoGym	✗	✓	✗	✗	✓	✓	✗
PyRecoGym	✗	✓	✗	✗	✓	✓	✗
Recsim	✗	✓	✗	✗	✓	✓	✗
MARS-Gym	✗	✓	✗	✗	✓	✓	✗
Accordion	✗	✓	✗	✗	✓	✓	✗
SIREN	✗	✓	✗	✗	✓	✓	✗
SimBa	✓	✓	✓	✓	✓	✓	✓

TABLE 5.2

Hyperparameters used for MF and IPSMF for both ML100K and Yahoo! R3 datasets

Model	Data	Learning rate	Latent Dimension	Regularization	Batch size
MF	ML-100K	0.013	8	0.0001	512
	Yahoo! R3	0.001	32	0.0001	256
IPSMF	ML-100K	0.0011	8	0.003	512
	Yahoo! R3	0.0011	16	0.003	512

CHAPTER 6

L-DYNAMICS: ADAPTIVE DYNAMIC DEBIASING STRATEGY FOR ITERATIVE RECOMMENDER SYSTEMS

Most debiasing strategies are static and do not take into account the iterative, and hence dynamic, nature of the recommender system. In this section, we propose a dynamic debiasing strategy in order to reduce the feedback loop effect through consecutive iterations. In this chapter, we present a **L**earnable **Dynamic** Debiasing **S**trategy For Iterative Recommender Systems (*L-DynamicS*) based on Trial-Offer Market Principles.

We start in Section 6.1 by presenting a theoretical motivation of the method by explaining the Trial-Offer (T-O) Markets in economic theory and showing how it relates to recommender systems. We then provide a deep dive of the debiasing strategy by detailing the different steps of the algorithm. Next, we present an empirical evaluation of L-DynamicS in Section 6.2. We use the SimBa framework, proposed earlier in Chapter 5, to test the effectiveness of L-DynamicS at providing accurate recommendation while maintaining a high coverage of the long tail items.

6.1 A New Learnable Dynamic Debiasing Strategy based on T-O Market Principles

To our knowledge, only one previous effort has explored a dynamic debiasing strategy to reduce the popularity bias in recommendation lists [121]. The idea is to rescale the predicted recommendations $\hat{r}_{ui}$ by a popularity coefficient Φ_i^α, where Φ_i is a popularity coefficient that indicates the proportion of number of ratings that item i has collected, and α is a debiasing degree. More formally, the rescaled ratings are computed using

$$\hat{r}_{ui}^{(scaled)} = \frac{\hat{r}_{ui}}{\Phi_i^\alpha} \tag{6.1}$$

and

$$\Phi_i = \frac{\sum_{u \in U} \mathbb{1}_{r_{ui} \neq 0}}{|U|} \tag{6.2}$$

In previous work [123], α was considered to be a static parameter that does not account for evolving popularity bias. Later, Zhu et.al [121] proposed a dynamic method to change the degree α as the iterations evolve, by increasing α by a step Δ in each iteration.

A main drawback of this method is that the dynamic step taken to adjust α is itself a hyperparameter and needs to be tuned. This is tricky, since in a real world scenario, it is hard to tune the model without risking to affect the user experience during the tuning process. For this reason, we propose a learnable dynamic debiasing strategy that learns the optimal debiasing degree α in each iteration. The main goal is to learn the value of the parameter α that will lead to a user's choice decision that is based on the quality of the recommendation *regardless* of the popularity of the item. One major difference between our method and most existing debiasing strategies including [121] is that it does not aim to suppress popular items for the benefit of unpopular ones. Instead, it aims at biasing the predicted rating *toward the highest quality items*. Quality in this context is measured in terms of relevance of the item to the user.

Therefore we propose a post-processing ranking method that rescales the model's predictions in order to push the system to converge to a stable state that is not dominated by only a few popular items, but rather items that are highly relevant to the user. Our method is inspired from economic theory, specifically the behavior of Trial-Offer (T-O) markets [120].

6.1.1 A theoretical analysis inspired by Trial-Offer markets

In this section, we provide some context and theory underpinning how we adapted the Trial-Offer Markets to the problem of dynamically debiasing recommendations. Many of the equations are based on results that have been introduced and proven in prior work, notably [120].

Definition 6.1.1. *a Trial-Offer Market is a market where the consumer can try a product before purchasing it.*

As described in Definition 6.1.1, T-O markets are markets where the user can try the product before purchasing it. For instance, many products nowadays launch with a free trial in order to give the consumer the opportunity to try the product. Other scenarios apply in this case like listening to a song snippet or watching a movie trailer in order to decide whether or not to purchase the song or watch the movie.

We assume that recommender systems can be modeled as a T-O market. In fact, most users read the reviews, watch a trailer, or inspect a product, while browsing the recommendation list, before deciding to purchase or watch the product. Also we can safely assume that deciding whether to provide feedback once a product is selected (tried) can be modeled as a T-O market. We summarize the analogies between a dynamic recommendation process and a T-O market in Table 6.1. Despite the similarities, one distinction of our work compared to T-O markets, is that the item's quality depends on the user's preferences; and hence we study the behavior from a user-centered perspective rather than an overall market perspective.

In a T-O market-inspired scenario, the click (analogous to a purchase in a market) decision can be modeled using

$$P(u \text{ clicks } i) = \frac{\hat{r}_{ui} v(\sigma_i)(\Phi_i)^\alpha}{\sum_{j \in I} \hat{r}_{uj} v(\sigma_j)(\Phi_j)^\alpha} \tag{6.3}$$

The idea is to study the evolution of the market when the social influence Φ_i (or the market shares) evolve through time. As some items get more clicks or views, these items naturally get more visibility and more influence. Therefore, it is natural to study how this visibility would affect the T-O market.

If we denote d_i^t as the number of views, purchases, or clicks on a given item i in time (or iteration) t, then the social influence of item i at iteration t can be defined as:

$$\Phi_i^t = \frac{d_i^t}{\sum_{j \in I} d_j^t} \tag{6.4}$$

The vector Φ_i^t thus defines the market share of the product i in terms of clicks, views, or purchases.

The theoretical results of the T-O Market model suggest that if the social influence degree $\alpha < 1$ then the market converges to a stable state, where the items with the highest quality will have a monopoly [120]. More formally, and inspired from [120], we can prove the following theorem for the dynamic recommender system setting, where we drop the iteration index t from the social influence Φ_i^t for convenience, for the sake of the proofs.

Theorem 6.1.2. *Given Φ as the popularity of the item, α as the popularity degree, a given item $i \in I$, and r_{ui} the relevance of item i to user u. Then, the popularity level Φ^* that achieves an equilibrium. An equilibrium is when the user's decision does not depend on the previous iteration of the system and only depends on the current state.*

$$\Phi_i^* = \frac{(v(\sigma_i) r_{ui})^{\frac{1}{1-\alpha}}}{\sum_{j \in I} (v(\sigma_j) r_{uj})^{\frac{1}{1-\alpha}}} \tag{6.5}$$

Proof. Considering $f(\Phi)$ =P(u clicks i) defined in Equation 6.3, we achieve equilibrium on Φ^* when

$$f(\Phi^*) = \Phi^* \tag{6.6}$$

Meaning that for a given social influence α, the market's social influence converges to a distribution correlated with the relevance of the item. This means that the social influence of the item will depend more on its quality (relevance to the user) and not its popularity.

To simplify our notation, we denote $v(\sigma_i) r_{ui}$ using r_{ui}^v. Therefore, the equilibrium is achieved when:

$$\frac{r_{uj}^v (\Phi_j^*)^\alpha}{\sum_{k \in I} r_{uk}^v (\Phi_k)^\alpha} = \Phi_j^*. \tag{6.7}$$

Dividing both sides by Φ_j^*, we get

$$r_{uj}^v (\Phi_j^*)^{\alpha - 1} = \sum_{j \in I} r_{uk}^v (\Phi_k)^\alpha \tag{6.8}$$

The same logic applies to any different item i, hence we get

$$r^v_{ui}(\Phi^*_i)^{\alpha-1} = \sum_{k\in I} r^v_{uk}(\Phi_k)^{\alpha}. \tag{6.9}$$

Therefore

$$r^v_{uj}(\Phi^*_j)^{\alpha-1} = r^v_{ui}(\Phi^*_i)^{\alpha-1} \tag{6.10}$$

By regrouping the terms and isolating Φ^*_i, we obtain

$$\Phi^*_j = (\frac{r^v_{ui}}{r^v_{uj}})^{\frac{1}{\alpha-1}}\Phi^*_i \tag{6.11}$$

summing both sides over all items $j \in I$ we get

$$1 = \sum_{j\in I}(\frac{r^v_{ui}}{r^v_{uj}})^{\frac{1}{\alpha-1}}\Phi^*_i \tag{6.12}$$

since as can be verified from (6.4), $\sum_j \Phi_j = 1$.

by rearranging the terms, we get our desired result, and without loss of generality, $\forall i \in I$ we have

$$\Phi^*_i = \frac{(v(\sigma_i)r_{ui})^{\frac{1}{1-\alpha}}}{\sum_{j\in I}(v(\sigma_j)r_{uj})^{\frac{1}{1-\alpha}}_j} \tag{6.13}$$

Therefore Φ^*_i is an equilibrium. □

What Theorem 6.1.2 states is that if the popularity distribution is correlated with the quantity $(v(\sigma_i)r_{ui})^{\frac{1}{1-\alpha}}$, then we reach an equilibrium. Equilibrium means that at each subsequent iteration, the distribution of the ratings will no longer be affected by the previous recommendations, as expressed in Equation 6.6.

We will use this result as an inspiration for our dynamic debiasing strategy. The idea is to rescale the predicted rating by $\Phi^{\hat{\alpha}}$, where $\hat{\alpha}$ is a parameter that will be **learned** to achieve an equilibrium in the system.

6.1.2 L-DynamicS: A learnable dynamic debiasing strategy for iterative recommender systems

Because it is important to dynamically debias the ratings produced by the recommendation model to take into account the feedback loop effect, a linear dynamic debiasing technique such as the one introduced in [121] will cause the system to diverge to a bias toward unpopular items, and this in turn will lead to a degradation of the accuracy of the model.

For this reason, we propose to learn an optimal debiasing social influence degree that aims at achieving an equilibrium within the system. An equilibrium in this case means that the evolution of the popularity of the items in the model is affected by the relevance of the item to the user and not the social influence of the item. In other words, the more relevant the item, the more clicks it should gather after many iterations.

To achieve this, the following assumptions need to be made:

- **(A 5.1)** We assume that for each item, the mean predicted rating, $\hat{r}_i = r^v_{ui}$, is a good estimate of the quality of the item. This is acceptable since the mean of all the ratings shows the tendency of how an item has been judged by all users, and hence is related to the quality of item. In a real world scenario, even the predicted rating can be affected by other biases such as conformity bias. We plan to extend this assumption to use less biased estimators to account for different types of bias in future work.

- **(A 5.2)** We assume that the social influence $\alpha = 1$ for all iterations. This means that the social influence is linearly correlated with the popularity. Other options would be to assume that the social influence has a quadratic or sublinear relationship with the popularity. To justify our choice, we refer to the work on T-O Markets that shows the effect of the social influence degree on the convergence of the market. [120] showed that if the social influence degree $\alpha < 1$, then the market will converge to a monopoly held by the highest quality item. On the other hand, if the social influence $\alpha \geq 1$ then the market diverges to a unique monopoly held by the most popular item. This is the

likely scenario in the recommender system case, as it has been shown that popularity bias gets worse after each iteration [121]. Therefore, the choice of $\alpha = 1$ is justified.

Using these assumptions, our debiasing strategy uses Theorem 6.1.2 in order to restore an equilibrium at every iteration, as follows:

Lemma 6.1.3. *Assuming A.5.1 and A.5.2 are satisfied and the framework for clicking an item follows Assumption 5.1.4, the debiasing strategy:*

$$\hat{r}^{scaled} = \frac{\hat{r}_i}{\Phi^{1-\bar{\alpha}}} \tag{6.14}$$

where $\bar{\alpha}$ is a social influence degree that satisfies $(\Phi^*)^{\bar{\alpha}} \propto \hat{r}$, achieves equilibrium in the current state of the dynamic recommender system.

Proof. Proving the Lemma uses the result from Theorem 6.1.2. In fact, if there exists $\bar{\beta}$ that satisfies the following relation

$$\Phi_i^* = \frac{(v(\sigma_i)r_{ui})^{\frac{1}{1-\bar{\alpha}}}}{\sum_{j\in I}(v(\sigma_j)r_{uj})^{\frac{1}{1-\bar{\beta}}}}$$

Then the system is at equilibrium. Now we use A.5.1 to substitute $v(\sigma_i)r_{ui}$ by $\hat{r}_i$. Therefore, The equilibrium is achieved if

$$\Phi_i^* = \frac{\hat{r}_i^{\frac{1}{1-\bar{\beta}}}}{\sum_{j\in I}\hat{r}_j^{\frac{1}{1-\bar{\beta}}}}$$

and since $\sum_{j\in I}\hat{r}_j^{\frac{1}{1-\bar{\beta}}}$ is a normalizing denominator independent of item i, it is sufficient to show that

$$\Phi_i^* \propto \hat{r}_i^{\frac{1}{1-\bar{\beta}}}$$

or that there exists $\bar{\alpha} = 1 - \bar{\beta}$ where

$$(\Phi_i^*)^{\bar{\alpha}} \propto \hat{r}_i$$

Now considering the choice model adopted in this framework and using A.5.2 to assert the present social influence degree defined in Assumption 5.1.4, we have

$$P(u \text{ clicks } i) = \frac{\hat{r}_i \Phi_i}{\sum_{j \in I} \hat{r}_{uj} \Phi_j}$$

Rescaling $\hat{r}$ by $\frac{\hat{r}_i}{\Phi_i^{1-\bar{\alpha}}}$ provides the following relation

$$P(u \text{ clicks } i) = \frac{\hat{r}_i \Phi_i^{\bar{\alpha}}}{\sum_{j \in I} \hat{r}_j \Phi_j^{\bar{\alpha}}}$$

Which is equivalent to the equilibrium equation (6.7) introduced in Theorem 6.1.2.

□

6.1.2.1 Learning the optimal debiasing degree

In order to learn an optimal debiasing degree $\bar{\alpha}$, we minimize the Jensen-Shannon Divergence [124] (JSD) between the distribution of the predicted relevance $\hat{r}$ and the popularity of the item, to make the distribution defined by $\Phi^{\bar{\alpha}}$ as close as possible to the distribution of $\hat{r}$ across all items. Therefore the loss function is given by

$$J(\alpha) = \frac{1}{2} KL(\hat{r}_i || \frac{(\hat{r}_i + \Phi^{\alpha})}{2}) + \frac{1}{2} KL(\Phi^{\alpha} || \frac{(\hat{r}_i + \Phi^{\alpha})}{2}), \tag{6.15}$$

with KL being the KL-Divergence [125] defined by

$$KL(P||Q) = \sum_x P(x) \log(\frac{P(x)}{Q(x)}) \tag{6.16}$$

We use gradient descent to learn an α that minimizes the loss function. The training data is defined by two sets $(\Phi_i)_{i \in I}$ and $(\hat{r}_i)_{i\ inI}$ that define the item popularity scores and mean predicted ratings, respectively. This results in the following update equation:

$$\alpha^{t+1} = \alpha^t - \lambda \frac{\mathrm{d}J}{\mathrm{d}\alpha}, \tag{6.17}$$

where

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha} = \Phi \times \log \Phi^{\alpha} \times \log \frac{\Phi}{\hat{r} + \Phi} \tag{6.18}$$

and λ is a learning rate. Once the loss converges, we use the learned α to rescale the predicted ratings. The full algorithm is presented in Algorithm 6.1

Algorithm 6.1 Learning the optimal debiasing degree in L-DynamicS

Input: Popularity scores Φ, Relevance scores $\hat{r}$, Learning rate λ, *iterations*, early stopping threshold ϵ
Output: Optimal debiasing degree $\bar{\alpha}$
initialize α^0
t=0
repeat
$J(\alpha^t) = \frac{1}{2}KL(\hat{r}_i||\frac{(\hat{r}_i+\Phi^{\alpha^t})}{2}) + \frac{1}{2}KL(\Phi^{\alpha^t}||\frac{(\hat{r}_i+\Phi^{\alpha^t})}{2})$ // Calculate Loss function
$\frac{\mathrm{d}J}{\mathrm{d}\alpha} = \Phi \times \log \Phi^{\alpha} \times \log \frac{\Phi}{\hat{r}+\Phi}$ // Calculate Gradient
$\alpha^{t+1} = \alpha^t - \lambda \frac{\mathrm{d}J}{\mathrm{d}\alpha}$ //perform update.
$t = t + 1$
until $|J(\alpha^t) - J(\alpha^{t+1})| < \epsilon$ OR $t ==$*iterations*
Return α

6.2 Dynamic Debiasing Evaluation

In this section, we evaluate our learnable dynamic debiasing strategy (L-DynamicS) proposed in Section 6.1.2, by using the SimBa simulation framework, that we proposed in Chapter 5, to conduct our experiments. We also use the Yahoo! R3 music dataset to assess the effectiveness of our model, because it provides unbiased real data as a seed for our simulation, which in turn leads to a well controlled introduced bias. We compare our proposed strategy, *L-DynamicS*, to the following post-processing debiasing baselines:

- **ϵ-greedy MAB [122]:** As shown in Section 5.2.7 , MAB is an effective post-processing strategy to add exploration to the results.
- **Static-Rescale [126]:** Rescale the predicted ratings $\hat{r}$ by $\frac{1}{\Phi}$. This is a static debiasing strategy
- **DScale [121]:** Dynamically rescale the predicted ratings by $\frac{1}{\Phi^{\alpha_t}}$ while increasing α_t after each iteration by a constant step size Δ

Figure 6.1 shows the results of all the methods in a dynamic experimental setting. We notice the effectiveness of our proposed strategy L-DynamicS, in recommending novel and relevant items compared to the Static-Rescale and DScale strategies. Although the previous strategies manage to reduce the bias of the system, the recommendations suffer from a considerable drop in relevance. The MAB strategy manages to provide relevant items

but we notice that the EPC metric starts falling quickly while the L-DynamicS strategy manages to outperform MAB on all other bias metrics. This confirms that learning an optimal debiasing degree α is effective for achieving a good trade-off between bias and relevance. We even notice an increase in the EPC metric after many iterations, which shows an increase in recommending relevant items even after many feedback loop iterations. The other post-processing methods show a different behavior as the recommendation quality deteriorates due to the feedback loop effect.

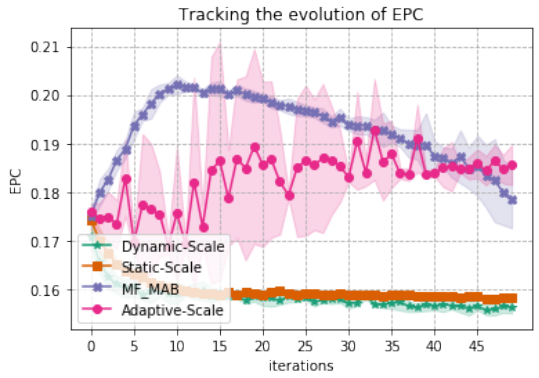
In Table 6.2, we list the EPC metric values at iteration 50 with the percent increase ΔEPC where

$$\Delta EPC = 100\frac{EPC_{t=50} - EPC_{t=0}}{EPC_{t=0}} \tag{6.19}$$

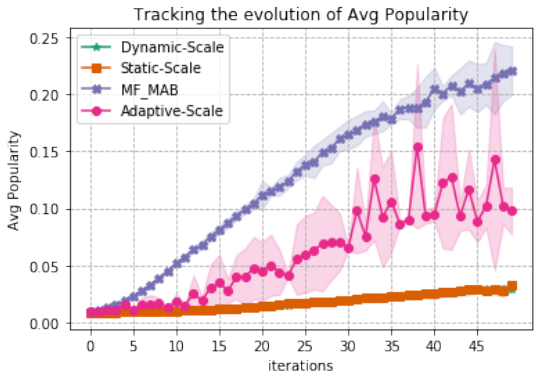
These results show that L-DynamicS achieves significantly better novelty and relevance levels even after running the simulation for 50 iterations. This behavior shows that by learning an adaptive debiasing step, we reduce the effect of the feedback loop on the system and we manage to increase the accuracy of the results **while** covering a more diverse set of items.

6.3 Summary

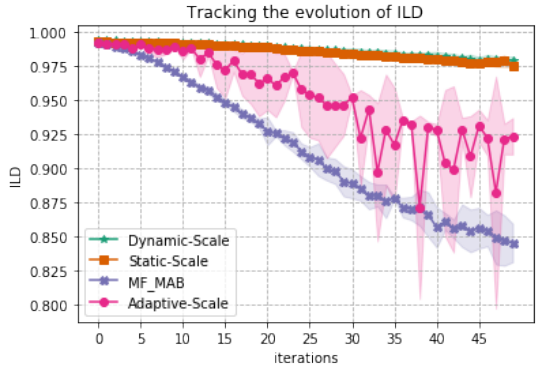
In this chapter, we proposed L-DynamicS: an Adaptive Dynamic Debias Strategy For Iterative Recommender Systems. We started by providing a theoretical background to motivate our approach, then we provide a detailed algorithm on learning an adaptive and optimal social influence degree to debias a given recommender system. Finally we provide experimental results to show the effectiveness of our method on a real word data and using our simulation framework SimBa.



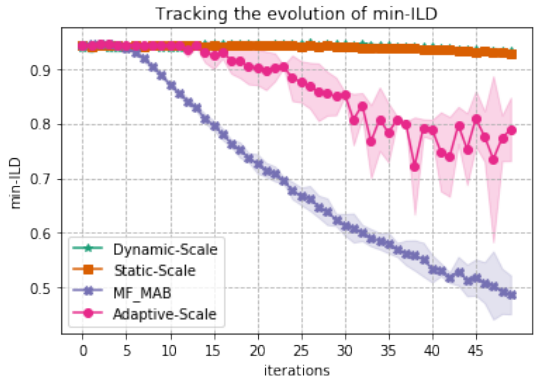
(a) Evolution of EPC for the Yahoo! R3 dataset



(b) Evolution of the Average Popularity for Yahoo! R3 dataset



(c) Evolution of the Intra-List Diversity for the Yahoo! R3 dataset



(d) Evolution of the Minimum Intra-List Diversity for the Yahoo! R3 dataset

Figure 6.1. The evolution of the bias metrics for the Yahoo! R3 dataset. We notice that MAB outperforms other metrics in terms of EPC. However it falls short in terms of diversity and popularity. Our Adaptive dynamic debiasing (L-DynamicS) outperforms the baseline methods by providing more accurate recommendations while covering more unpopular items. Hence, using a static or dynamic debiasing obtains better debiasing but only at the expense of significantly worse relevance in the results.

TABLE 6.1

Analogies between a T-O market and a dynamic recommender system scenario

T-O market	**Recommender system**
Try the product then make a purchase decision	Browse a recommendation list then make a decision to click on an item
Social influence of products	Popularity of items
Visibility of products	Rank of the item in the recommendation list
Quality of the product	Relevance of the item to the user

TABLE 6.2

Comparison of the EPC level at iteration 50 and percent increase of EPC for different debiasing strategies

Method	EPC	ΔEPC (%)
Dynamic-Scale	0.156205	-8.5499
Static-Scale	0.158359	-9.1542
MAB	0.178609	1.9452
Adaptive scale	**0.185565**	**5.4622**

CHAPTER 7

CONCLUSION

The main goal of this dissertation is to study feedback loops and bias in recommender systems from three main perspectives: modeling feedback loops, debiasing recommendation systems, and auditing recommendation systems for bias. In Chapter 3, we proposed a new deep learning architecture that we called the Transformer-Promoter network, a Transformer-based architecture for Collaborative Filtering algorithms, that adds regularization in its attention mechanism in order to promote more diverse recommendations. We use the distance between the centroids of the neighbors' items (users) to the given user (item) in order to regularize the attention weights into attending to more diverse neighbors. The Transformer-Promoter network shows improved results in terms of accuracy and diversity over other competitive baseline methods such as Neural Matrix Factorization and Matrix Factorization. We also evaluated the quality of the learned embedding and we noticed that the Transformer-Promoter network learns diverse embeddings for similar items while keeping the convex hull consistent with the items' neighborhood.

The main limitation of the Transformer-Promoter network is that it is affected by the cold start problem. In fact, without previous interactions, we cannot construct the contextual neighborhood needed for the Attention unit to work. One might solve this issue by including external features for users and items, within a hybrid recommendation paradigm. In fact, user and item similarity can be computed based on these features. Although user features are expensive to acquire due to privacy concerns, item features can be extracted and can enrich the item embeddings and overcome the cold start problem. In our work, we did not experiment with implicit feedback as we tried to control for the extra bias coming from the techniques used in this case, such as negative sampling. Implicit

feedback settings are very important as the data provided by the users, in general, are in the form of clicks, views, or interactions that cannot always be translated to explicit feedback.

In Chapter 4, we first presented a theoretical study describing the evolution of the feedback loop in a collaborative filtering recommender system. We modeled the user discovery as a Martingale difference and showed that after a few iterations, the user discovery converges to zero with an exponentially decaying rate. These results show that a collaborative filtering system, under a few assumptions, will result in a filter bubble. We empirically showed the validity of our results using a simulated feedback loop. We also relaxed the assumptions that we made to derive our proofs to show the effect of the exploration methods on the feedback loop. We showed that even under basic exploration strategies, the model still converges to a static state where the user is trapped in a filter bubble. These findings could inspire new theoretically rooted methods to counteract the negative effect of feedback loops.

The limitations of our theoretical modeling of user discovery come mainly from the assumptions that we have stated. In fact, the user, in real life, is exposed to different external influences, including from different recommender system platforms. Although some of these external exposures may be related either directly (for instance through external data, cookies, etc), or indirectly (through similar social circles), studying an isolated recommender system is limited and needs to be extended to a more general framework.

In Chapter 5, we proposed an evaluation methodology and tool consisting of a modular simulation framework that mimics the iterative behavior of recommender systems. Our simulation framework can be used to quantify the bias in a given recommender system. Therefore it can be used as an auditing tool to detect and control for different sources of bias.

Our simulation study opens several directions for future work. One possibility is to experiment with reinforcement learning-based approaches (beyond Multi-Armed Bandits approaches) in order to evaluate their sequential behavior. Future work can also include more sophisticated user models in order to get closer to a real world scenario.

In Chapter 6 proposed a learnable social influence dynamic debiasing strategy (L-DynamicS) that is able to learn an optimal debiasing degree α. This adaptive dynamic debiasing strategy yields more relevant results without being affected by the feedback loop, and while still being robust to bias. Our method has the advantage that it can be used with different recommendation models since it is model agnostic. It also can be extended to allow learning the bias degree while learning the recommendation model as a multi-task learning step.

With the increasing growth in the use of advanced data-driven machine learning models, tools to assess potential bias and decrease the resulting unfairness are becoming essential to reduce the inevitable harms that could be caused by AI systems. This dissertation paves the way toward advancing the state of the art in the field of fairness in AI and more specifically in recommender systems. Because recommender systems can influence human discovery, behavior, and decision making in diverse domains, it is essential to be able to understand, assess and mitigate their bias.

To help the research and practice community in studying recommender system fairness, all the tools developed to model, debias, and evaluate recommender systems are made available to the public as open source software libraries [1].

[1] https://github.com/samikhenissi/TheoretUserModeling

REFERENCES

[1] "Gartner says nearly half of cios are planning to deploy artificial intelligence," .

[2] Yifan Hu, Yehuda Koren, and Chris Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 263–272.

[3] Nishigandha Karbhari, Asmita Deshmukh, and Vinayak D. Shinde, "Recommendation system using content filtering: A case study for college campus placement," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017, pp. 963–965.

[4] Robin Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[5] Elizabeth Gómez, Ludovico Boratto, and Maria Salamó, "Provider fairness across continents in collaborative recommender systems," *Information Processing Management*, vol. 59, no. 1, pp. 102719, 2022.

[6] Ricardo Baeza-Yates, "Bias on the web," *Commun. ACM*, vol. 61, no. 6, pp. 54–61, May 2018.

[7] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, "Item-based collaborative filtering recommendation algorithms," *Proceedings of ACM World Wide Web Conference*, vol. 1, 08 2001.

[8] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, Republic and Canton of Geneva, CHE, 2017, WWW '17, p. 173–182, International World Wide Web Conferences Steering Committee.

[10] Xiaoning JIN Guijuan ZHANG, Yang LIU, "A survey of autoencoder-based recommender systems," *Frontiers of Computer Science*, vol. 14, no. 2, pp. 430, 2020.

[11] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui, "Graph neural networks in recommender systems: A survey," *CoRR*, vol. abs/2011.02260, 2020.

[12] Ray Jiang, Silvia Chiappa, Tor Lattimore, András György, and Pushmeet Kohli, "Degenerate feedback loops in recommender systems," *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, Jan. 2019.

[13] Matevž Kunaver and Tomaž Požrl, "Diversity in recommender systems – a survey," *Knowledge-Based Systems*, vol. 123, pp. 154–162, 2017.

[14] Harald Steck, "Item popularity and recommendation accuracy," in *Proceedings of the Fifth ACM Conference on Recommender Systems*, New York, NY, USA, 2011, RecSys '11, p. 125–132, Association for Computing Machinery.

[15] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 240–248, Association for Computing Machinery.

[16] Sami Khenissi, Boujelbene Mariem, and Olfa Nasraoui, "Theoretical modeling of the iterative properties of user discovery in a collaborative filtering recommender system," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 348–357, Association for Computing Machinery.

[17] Wenlong Sun, Sami Khenissi, Olfa Nasraoui, and Patrick Shafto, *Debiasing the Human-Recommender System Feedback Loop in Collaborative Filtering*, p. 645–651, Association for Computing Machinery, New York, NY, USA, 2019.

[18] Dimitrios Bountouridis, Jaron Harambam, Mykola Makhortykh, Mónica Marrero, Nava Tintarev, and Claudia Hauff, "Siren: A simulation framework for understanding the effects of recommender systems in online news environments," New York, NY, USA, 2019, FAT* '19, p. 150–159, Association for Computing Machinery.

[19] Shuo Zhang, "Measuring algorithmic bias in job recommender systems: An audit study approach," 2021.

[20] Michael D Ekstrand, John T Riedl, and Joseph A Konstan, *Collaborative filtering recommender systems*, Now Publishers Inc, 2011.

[21] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro, "Content-based recommender systems: State of the art and trends," *Recommender systems handbook*, pp. 73–105, 2011.

[22] Wei Chen, Tie-yan Liu, Yanyan Lan, Zhi-ming Ma, and Hang Li, "Ranking measures and loss functions in learning to rank," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds. 2009, vol. 22, Curran Associates, Inc.

[23] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai, "Learning tree-based deep model for recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1079–1088.

[24] Deuk Hee Park, Hyea Kyeong Kim, Il Young Choi, and Jae Kyeong Kim, "A literature review and classification of recommender systems research," *Expert systems with applications*, vol. 39, no. 11, pp. 10059–10072, 2012.

[25] Tianqi Chen and Carlos Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2016, KDD '16, p. 785–794, Association for Computing Machinery.

[26] Benedikt Schifferer, Gilberto Titericz, Chris Deotte, Christof Henkel, Kazuki Onodera, Jiwei Liu, Bojan Tunguz, Even Oldridge, Gabriel De Souza Pereira Moreira, and Ahmet Erdem, "Gpu accelerated feature engineering and training for recommender systems," in *Proceedings of the Recommender Systems Challenge 2020*, New York, NY, USA, 2020, RecSysChallenge '20, p. 16–23, Association for Computing Machinery.

[27] Longteng Xu, Jiwei Liu, and Yu Gu, "A recommendation system based on extreme gradient boosting classifier," in *2018 10th International Conference on Modelling, Identification and Control (ICMIC)*, 2018, pp. 1–5.

[28] Christopher J. C. Burges, Krysta M. Svore, Paul N. Bennett, Andrzej Pastusiak, and Qiang Wu, "Learning to rank using an ensemble of lambda-gradient models," in *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14*. 2010, YLRC'10, p. 25–35, JMLR.org.

[29] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao, "Learning to rank by optimizing ndcg measure," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds. 2009, vol. 22, Curran Associates, Inc.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, vol. 30, pp. 5998–6008, Curran Associates, Inc.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[32] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, feb 2019.

[33] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tyler Sax, Lanbo Zhang, Aamir Mansawala, Shulin Yang, Bradley Turnbull, and Junshuo Liao, "Improving deep learning for airbnb search," 2020.

[34] Michael Pulis and Josef Bajada, *Siamese Neural Networks for Content-Based Cold-Start Music Recommendation.*, p. 719–723, Association for Computing Machinery, New York, NY, USA, 2021.

[35] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec, "Pinnersage: Multi-modal user embedding framework for recommendations at pinterest," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, New York, NY, USA, 2020, KDD '20, p. 2311–2320, Association for Computing Machinery.

[36] Sebastian Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[37] Paul Covington, Jay Adams, and Emre Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016, RecSys '16, p. 191–198, Association for Computing Machinery.

[38] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims, "Recommendations as treatments: Debiasing learning and evaluation," *CoRR*, vol. abs/1602.05352, 2016.

[39] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi, "Doubly robust joint learning for recommendation on data missing not at random," Long Beach, California, USA,

09–15 Jun 2019, vol. 97 of *Proceedings of Machine Learning Research*, pp. 6638–6647, PMLR.

[40] Andriy Mnih and Russ R Salakhutdinov, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. 2008, vol. 20, Curran Associates, Inc.

[41] Prem Gopalan, Jake M. Hofman, and David M. Blei, "Scalable recommendation with poisson factorization," 2014.

[42] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, Arlington, Virginia, USA, 2009, UAI '09, p. 452–461, AUAI Press.

[43] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan, "Rethinking neural vs. matrix-factorization collaborative filtering: the theoretical perspectives," in *Proceedings of the 38th International Conference on Machine Learning*, Marina Meila and Tong Zhang, Eds. 18–24 Jul 2021, vol. 139 of *Proceedings of Machine Learning Research*, pp. 11514–11524, PMLR.

[44] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web*, New York, NY, USA, 2015, WWW '15 Companion, p. 111–112, Association for Computing Machinery.

[45] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara, "Variational autoencoders for collaborative filtering," 2018.

[46] Adji B. Dieng, Yoon Kim, Alexander M. Rush, and David M. Blei, "Avoiding latent variable collapse with generative skip models," 2019.

[47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, June 2019, pp. 4171–4186, Association for Computational Linguistics.

[48] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, "Language models are unsupervised multitask learners," 2019.

[49] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei, "Language models are few-shot learners," 2020.

[50] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020.

[51] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," 2019.

[52] Wang-Cheng Kang and Julian J. McAuley, "Self-attentive sequential recommendation," *CoRR*, vol. abs/1808.09781, 2018.

[53] Jing Lin, Weike Pan, and Zhong Ming, "Fissa: Fusing item similarity models with self-attention networks for sequential recommendation," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 130–139, Association for Computing Machinery.

[54] Jin Peng Zhou, Zhaoyue Cheng, Felipe Perez, and Maksims Volkovs, "Tafa: Two-headed attention fused autoencoder for context-aware recommendations," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 338–347, Association for Computing Machinery.

[55] Jibing Gong, Shen Wang, Jinlong Wang, Wenzheng Feng, Hao Peng, Jie Tang, and Philip S. Yu, *Attentional Graph Convolutional Networks for Knowledge Concept Recommendation in MOOCs in a Heterogeneous View*, p. 79–88, Association for Computing Machinery, New York, NY, USA, 2020.

[56] Sebastian Hofstätter, Hamed Zamani, Bhaskar Mitra, Nick Craswell, and Allan Hanbury, *Local Self-Attention over Long Text for Efficient Document Retrieval*, p. 2021–2024, Association for Computing Machinery, New York, NY, USA, 2020.

[57] Weinan Xu, Hengxu He, Minshi Tan, Yunming Li, Jun Lang, and Dongbai Guo, "Deep interest with hierarchical attention network for click-through rate prediction," 2020.

[58] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang, "Solving the apparent diversity-accuracy dilemma of recommender systems," *Proceedings of the National Academy of Sciences*, vol. 107, no. 10, pp. 4511–4515, 2010.

[59] Laming Chen, Guoxin Zhang, and Hanning Zhou, "Improving the diversity of top-n recommendation via determinantal point process," 09 2017.

[60] Lijing Qin and Xiaoyan Zhu, "Promoting diversity in recommendation by entropy regularizer," 2013.

[61] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong, "Learning to recommend accurate and diverse items," in *Proceedings of the 26th International Conference on World Wide Web - WWW '17*. 2017, ACM Press.

[62] Anisio Lacerda, "Multi-objective ranked bandits for recommender systems," *Neurocomputing*, vol. 246, 02 2017.

[63] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson, and Rishabh Mehrotra, "Explore, exploit, and explain: Personalizing explainable recommendations with bandits," in *Proceedings of the 12th ACM Conference on Recommender Systems*, New York, NY, USA, 2018, RecSys '18, p. 31–39, Association for Computing Machinery.

[64] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims, "Learning diverse rankings with multi-armed bandits," in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, ICML '08, p. 784–791, Association for Computing Machinery.

[65] Chang Li, Haoyun Feng, and Maarten de Rijke, "Cascading hybrid bandits: Online learning to rank for relevance and diversity," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 33–42, Association for Computing Machinery.

[66] Javier Parapar and Filip Radlinski, "Diverse user preference elicitation with multi-armed bandits," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, New York, NY, USA, 2021, WSDM '21, p. 130–138, Association for Computing Machinery.

[67] Andrea Barraza-Urbina, Benjamin Heitmann, Conor Hayes, and Angela Carrillo Ramos, "Xplodiv: An exploitation-exploration aware diversification approach for recommender systems," in *FLAIRS Conference*, 2015.

[68] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li, "Drn: A deep reinforcement learning framework for news recommendation.," in *WWW*, Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, Eds. 2018, pp. 167–176, ACM.

[69] Yong Liu, Yinan Zhang, Qiong Wu, Chunyan Miao, Lizhen Cui, Binqiang Zhao, Yin Zhao, and Lu Guan, "Diversity-promoting deep reinforcement learning for interactive recommendation," 2019.

[70] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He, "Bias and debias in recommender system: A survey and future directions," 2020.

[71] Olivier Jeunen, David Rohde, Flavian Vasile, and Martin Bompaire, "Joint policy-value learning for recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, New York, NY, USA, 2020, KDD '20, p. 1223–1233, Association for Computing Machinery.

[72] Mahsa Badami, Olfa Nasraoui, and Patrick Shafto, "Prcp: Pre-recommendation counter-polarization," in *Proceedings Of the Knowledge Discovery and Information Retrieval conference, Seville, Spain*, 2018.

[73] Natali Helberger, Kari Karppinen, and Lucia D'Acunto, "Exposure diversity as a design principle for recommender systems," *Information, Communication & Society*, vol. 21, no. 2, pp. 191–207, 2018.

[74] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher, "The unfairness of popularity bias in recommendation," 08 2019.

[75] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher, "Controlling popularity bias in learning-to-rank recommendation," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, New York, NY, USA, 2017, RecSys '17, p. 42–46, Association for Computing Machinery.

[76] D. Kiswanto, D. Nurjanah, and R. Rismala, "Fairness aware regularization on a learning-to-rank recommender system for controlling popularity bias in e-commerce domain," in *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*, Oct. 2018, pp. 16–21.

[77] Himan Abdollahpouri, Robin D. Burke, and Bamshad Mobasher, "Managing popularity bias in recommender systems with personalized re-ranking," *ArXiv*, vol. abs/1901.07555, 2019.

[78] Dietmar Jannach, Lukas Lerche, Iman Kamehkhosh, and Michael Jugovac, "What recommenders recommend: an analysis of recommendation biases and possible countermeasures," *User Modeling and User-Adapted Interaction*, vol. 25, no. 5, pp. 427–491, July 2015.

[79] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin, "Unbiased offline recommender evaluation for missing-not-at-random implicit feedback," in *Proceedings of the 12th ACM Conference on Recommender Systems*, New York, NY, USA, 2018, RecSys '18, p. 279–287, Association for Computing Machinery.

[80] Wei Ma and George H Chen, "Missing not at random in matrix completion: The effectiveness of estimating missingness probabilities under a low nuclear norm assumption," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 14871–14880. Curran Associates, Inc., 2019.

[81] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft, "Unbiased learning to rank with unbiased propensity estimation," in *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval*, New York, NY, USA, 2018, SIGIR '18, p. 385–394, Association for Computing Machinery.

[82] Sami Khenissi and Olfa Nasraoui, "Modeling and counteracting exposure bias in recommender systems," 2020.

[83] Menghan Wang, Mingming Gong, Xiaolin Zheng, and Kun Zhang, "Modeling dynamic missingness of implicit feedback for recommendation," Red Hook, NY, USA, 2018, NIPS'18, p. 6670–6679, Curran Associates Inc.

[84] Jonathan P. Epperlein, Sergiy Zhuk, and Robert Shorten, "Recovering markov models from closed-loop data," *Automatica*, vol. 103, pp. 116–125, 2019.

[85] Olfa Nasraoui and Patrick Shafto, "Human-algorithm interaction biases in the big data cycle: A markov chain iterated learning framework," *arXiv preprint arXiv:1608.07895*, 2016.

[86] Ayan Sinha, David F. Gleich, and Karthik Ramani, "Deconvolving feedback loops in recommender systems," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, Eds., 2016, pp. 3243–3251.

[87] Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini, "The echo chamber effect on social media," *Proceedings of the National Academy of Sciences*, vol. 118, no. 9, 2021.

[88] Pablo Barberá, *Social Media, Echo Chambers, and Political Polarization*, p. 34–55, SSRC Anxieties of Democracy. Cambridge University Press, 2020.

[89] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke, *Feedback Loop and Bias Amplification in Recommender Systems*, p. 2145–2148, Association for Computing Machinery, New York, NY, USA, 2020.

[90] Masoud Mansoury, "Understanding and mitigating multi-sided exposure bias in recommender systems," 2021.

[91] Andres Ferraro, Xavier Serra, and Christine Bauer, *Break the Loop: Gender Imbalance in Music Recommenders*, p. 249–254, Association for Computing Machinery, New York, NY, USA, 2021.

[92] Khalil Damak, Sami Khenissi, and Olfa Nasraoui, *Debiased Explainable Pairwise Ranking from Implicit Feedback*, p. 321–331, Association for Computing Machinery, New York, NY, USA, 2021.

[93] Olivier Jeunen, Koen Verstrepen, and Bart Goethals, "Fair offline evaluation methodologies for implicit-feedback recommender systems with mnar data," 2018.

[94] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof, "Keeping dataset biases out of the simulation: A debiased simulator for reinforcement learning based recommender systems," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 190–199, Association for Computing Machinery.

[95] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier, "Recsim: A configurable simulation platform for recommender systems," 2019.

[96] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou, "Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising," *arXiv preprint arXiv:1808.00720*, 2018.

[97] Bichen Shi, Makbule Gulcin Ozsoy, Neil Hurley, Barry Smyth, Elias Z. Tragos, James Geraci, and Aonghus Lawlor, "Pyrecgym: A reinforcement learning gym for recommender systems," in *Proceedings of the 13th ACM Conference on Recommender Systems*, New York, NY, USA, 2019, RecSys '19, p. 491–495, Association for Computing Machinery.

[98] Marlesson R. O. Santana, Luckeciano C. Melo, Fernando H. F. Camargo, Bruno Brandão, Anderson Soares, Renan M. Oliveira, and Sandor Caetano, "Mars-gym: A gym framework to model, train, and evaluate recommender systems for marketplaces," 2020.

[99] Shuo Zhang and Krisztian Balog, "Evaluating conversational recommender systems via user simulation," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2020.

[100] Francesco Fabbri, Maria Luisa Croci, Francesco Bonchi, and Carlos Castillo, "Exposure inequality in people recommender systems: The long-term effects," *CoRR*, vol. abs/2112.08237, 2021.

[101] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed Chi, "Top-k off-policy correction for a reinforce recommender system," 2019.

[102] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke, "Feedback loop and bias amplification in recommender systems," 2020.

[103] Sirui Yao, Yoni Halpern, Nithum Thain, Xuezhi Wang, Kang Lee, Flavien Prost, Ed H. Chi, Jilin Chen, and Alex Beutel, "Measuring recommender system effects with simulated users," 2021.

[104] James McInerney, Ehtsham Elahi, Justin Basilico, Yves Raimond, and Tony Jebara, *Accordion: A Trainable Simulator for Long-Term Interactive Systems*, p. 102–113, Association for Computing Machinery, New York, NY, USA, 2021.

[105] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," 2020.

[106] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng, "Are we evaluating rigorously? benchmarking recommendation for reproducible evaluation and fair comparison," in *Fourteenth ACM Conference on Recommender Systems*, New York, NY, USA, 2020, RecSys '20, p. 23–32, Association for Computing Machinery.

[107] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. 2011, vol. 24, pp. 2546–2554, Curran Associates, Inc.

[108] Arda Antikacioglu, Tanvi Bajpai, and R. Ravi, "A new system-wide diversity measure for recommendations with efficient algorithms," 2019.

[109] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma, "How good your recommender system is? a survey on evaluations in recommendation," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 5, pp. 813–831, Dec. 2017.

[110] Elvin Isufi, Matteo Pocchiari, and Alan Hanjalic, "Accuracy-diversity trade-off in recommender systems via graph convolutions," *Information Processing Management*, vol. 58, no. 2, pp. 102459, 2021.

[111] Arda Antikacioglu and R. Ravi, "Post processing recommender systems for diversity," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2017, KDD '17, p. 707–716, Association for Computing Machinery.

[112] Laurens van der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[113] Saúl Vargas and Pablo Castells, "Rank and relevance in novelty and diversity metrics for recommender systems," in *Proceedings of the Fifth ACM Conference on Recommender Systems*, New York, NY, USA, 2011, RecSys '11, p. 109–116, Association for Computing Machinery.

[114] "Robot learning, edited by jonathan h. connell and sridhar mahadevan, kluwer, boston, 1993/1997, xii 240pp., isbn 0-7923-9365-1 (hardback, 218.00 guilders, 120.00, £89.95).," *Robotica*, vol. 17, no. 2, pp. 229–235, 1999.

[115] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua, "Neural collaborative filtering," 2017.

[116] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[117] Erion Çano and Maurizio Morisio, "Hybrid recommender systems: A systematic literature review," *CoRR*, vol. abs/1901.03888, 2019.

[118] Yichao Wang, Xiangyu Zhang, Zhirong Liu, Zhenhua Dong, Xinhua Feng, Ruiming Tang, and Xiuqiang He, "Personalized re-ranking for improving diversity in live recommender systems," *CoRR*, vol. abs/2004.06390, 2020.

[119] Mohammad Mehdi Afsar, Trafford Crump, and Behrouz H. Far, "Reinforcement learning based recommender systems: A survey," *CoRR*, vol. abs/2101.06286, 2021.

[120] Felipe Maldonado, Pascal Van Hentenryck, Gerardo Berbeglia, and Franco Berbeglia, "Popularity signals in trial-offer markets with social influence and position bias," *European Journal of Operational Research*, vol. 266, pp. 775–793, 04 2018.

[121] Ziwei Zhu, Yun He, Xing Zhao, and James Caverlee, *Popularity Bias in Dynamic Recommendation*, p. 2439–2449, Association for Computing Machinery, New York, NY, USA, 2021.

[122] Gangan Elena, Kudus Milos, and Ilyushin Eugene, "Survey of multiarmed bandit algorithms applied to recommendation systems," *International Journal of Open Information Technologies*, vol. 9, no. 4, pp. 12–27, 2021.

[123] Harald Steck, "Collaborative filtering via high-dimensional regression," 2019.

[124] Christopher D. Manning and Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, The MIT Press, Cambridge, Massachusetts, 1999.

[125] James M. Joyce, *Kullback-Leibler Divergence*, pp. 720–722, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[126] Harald Steck, "Collaborative filtering via high-dimensional regression," *CoRR*, vol. abs/1904.13033, 2019.

CURRICULUM VITAE

NAME: Sami Khenissi

ADDRESS: Computer Science and Engineering Department
J.B Speed School of Engineering
University of Louisville
Louisville, KY 40292
United States of America.

EDUCATION: Ph.D., Computer Science & Engineering, May 2022
University of Louisville, *Louisville, KY, USA.*
M.Sc. in Computer Science, May 2019
University of Louisville, *Louisville, KY, USA.*
B.Eng in Applied Mathematics, September 2017
Ecole Polytechnique de Tunisie, *Tunisia*

WORK EXPERIENCE: **Graduate Assistant**, *University of Louisville*, KY, USA, 2017 - 2022
Research scientist intern, *Bloomberg*, New York, USA, May - August 2021

AWARDS: **Dean Citation for academic excellence** , 2019, 2022
University Fellowship, 2020
Highest cumulative scholastic standing in the departmental Master of Science program, 2019

PUBLICATIONS:

1. Wenlong Sun, **Sami Khenissi**, Olfa Nasraoui, and Patrick Shafto. 2019. Debiasing the Human-Recommender System Feedback Loop in Collaborative Filtering. In International workshop on Augmenting Intelligence with Bias-Aware Humans-in-the-Loop, World Wide Web Conference, 2019.

2. **Sami Khenissi**, Boujelbene Mariem, and Olfa Nasraoui. 2020. Theoretical Modeling of the Iterative Properties of User Discovery in a Collaborative Filtering Recommender System. In Fourteenth ACM Conference on Recommender Systems (RecSys '20). Association for Computing Machinery, New York, NY, USA, 348–357.

3.Khalil Damak, **Sami Khenissi**, and Olfa Nasraoui. 2021. Debiased Explainable Pairwise Ranking from Implicit Feedback. Fifteenth ACM Conference on Recommender Systems. Association for Computing Machinery, New York, NY, USA, 321–331

4.**Sami Khenissi**, Mariem Boujelbene, Khalil Damak, Olfa Nasraoui. 2022. A New Attention Framework for Promoting Diverse Recommendations. Under Review. ACM SIGKDD 2022