Modeling Heterogeneous Users Behaviors in Online Systems

A Dissertation

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Data Science

by

Thanh Tran

September 03, 2020

APPROVED:

Professor Kyumin Lee Worcester Polytechnic Institute Advisor	Professor Xiangnan Kong Worcester Polytechnic Institute Committee Member
Professor Randy C. Paffenroth Worcester Polytechnic Institute Committee Member	Professor Dongwon Lee Penn State University External Committee Member
Professor Elke A. Rundensteiner Worcester Polytechnic Institute Program Director	

Abstract

Many online systems such as e-commerce, music/video streaming platforms have been proliferating in recent decades, creating dramatic changes in people's shopping experiences by providing accessibility to incredible volumes of products, and enabling millions of users to sell/purchase online commodities. In such systems, understanding behavior of both product sellers and customers, two main objects of the online systems, is important to the systems' prosperity. Thus, this dissertation makes three unique contributions as followings:

First, we focus on understanding and characterizing the product delivery activities of the product owners. This task has played an essential role in maintaining not only the trust between the consumers and the product owners but also the trust between these two objects and the platform providers. Unfortunately, in the literature, little is known to address the problem. In this direction, we extract novel features that reveal factors, which influence to the product delivery phase of the product owners. As a result, we build predictive models for on-time product delivery identification and delivery duration time estimation in the crowdfunding platforms.

Second, we investigate the problem of modeling consumer behaviors with global constraints. The problem is crucial in many applications like basket-based shopping platforms, video/music streaming services (i.e. Spotify, YouTube, *etc.*). This is mainly because a consumer preference is often decided by the

general taste of all products she/he preferred so far in her/his current session. In this line, we present a Matrix Factorization (MF) based recommender with several constraints on global similar product embeddings and global similar consumer embeddings. Due to the fact that MF based methods are intrinsic to a linear nature and dot product operator in MF based methods do not convey the crucial triangle inequality, we further proposed three novel metric learning-based neural recommenders to encode complex preferences of customers over products better. Moreover, we improve the robustness of our models by applying adversarial personalized ranking and customizing it with a flexible noise.

Finally, we study the task of modeling consumer behaviors with both long-term and short-term interest dependencies. In many e-commerce platforms like Amazon, Netflix and Yelp, encoding a consumer long-term preference dependency based on all of her interacted products so far is not enough. The main reason is that her preferred next product can have a strong correlation with her current interest, which is reflected by her recently preferred items. To address the task, we present signed distance-based neural recommenders. Furthermore, we go beyond the Euclidean representation space and present our Quaternion-based recommenders that introduce the benefits of Quaternion space in modeling the consumer preferences with both long-term and short-term dependencies.

Dedication

To my wife, my son, and my parents

Acknowledgements

First of all, I would like to thank my wife, Chung Hoang, for leaving behind all her great successes in her profession in our country to be here in the U.S with me, to support me to achieve at this point of my profession, to encourage me whenever I failed to implement my ideas or when my ideas did not go well. Together with my son Andrew, they are my biggest motivation to keep me working hard in this long, challenging, but valuable Ph.D. journey. I am also grateful to have my best parents – Hai Yen Thi Nguyen and Thach Ngoc Tran – who inspired me to pursue a doctorate program. Together with my parents in law, Thuy Do and Vien Hoang, all of them have always been being by my side to help me. Even though living in another half of the earth, when my son Andrew was born, my mother and my mother in law travelled to the U.S to cheer us up and to support us, allowing me to focus on my work during that hard time, and I am grateful. I also feel blessed when having my lovely American parents – Christine Hult and Nathan Hult. You have been sharing the highs and the lows of my Ph.D. adventure since I started. Your emotional and practical support has nourished my open mindedness, work ethic, and creative problem solving. I love all my brothers and sisters and thank all of you for your tremendous supports during my Ph.D. life.

I sincerely thank my advisor, Prof. Kyumin Lee, for supporting me whenever I need, providing me with multiple inspiring and valuable discussions and comments to all my publications. Prof. Lee is always calm to wait for my academic maturity and to stay together with me to fight against the publication deadlines. More importantly, Prof. Lee has been giving me so much life experience and advice since my very first days coming to the U.S with limited experience and a culture shock. Without his help, I would certainly not be where I am now in my personal and professional development.

I would like to thank all my committee members, Prof. Dongwon Lee, Prof. Xiangnan Kong, Prof. Randy Paffenroth, for providing me with a lot of fruitful comments and helpful discussions on my research whenever I got stuck, and for spending your precious time reading my publications and this dissertation. Especially, I am very grateful to Prof. Dongwon Lee for helping me formulate my ideas much better in my first days and my first publication working in recommendation systems, and Prof. Xiangnan Kong for his detailed attention on theory and motivation of my recent publications to improve its quality to the highest point.

I thank Prof. Elke A. Rundensteiner and the Data Science program of Worcester Polytechnic Institute for providing me with a lot of training classes, great programs and research colloquiums where I can learn a lot of novel ideas from other researchers, as well as providing me so many competitions to join in and learn from my colleagues. Those fundamental materials are undoubtedly must-have equipment for my current and future professional development.

I would like to thank all my collaborators, Di You, Nguyen Vo, Renee Sweeney, Yiming Liao, Xinyue Liu, Prudhvi Ratna Badri Satya, for supporting me with their brilliant ideas and productive collaborations. They elaborated and polished many unclear important points in many of my publications, and strengthened my publications and knowledge in many aspects. It was my great pleasure to have opportunities to collaborate with them.

I also thank members of our research group, Guanyi Mou, Qian Wang, Yichuan Li and all other members in the DSRG group for sharing with me their researches and helping me to consolidate my research knowledge gap. They are always there and are so friendly to answer all kinds of questions I have.

I would like to express thanks to Yifan Hu and other members of the Data Science team for giving me a research opportunity at the Yahoo! Research Lab.

Last but not least, I would like to thank Mary Racicot, our conscientious administrative assistant of the Data Science Program, for helping me out with all administrative works.

Publications

12. Thanh Tran, Yifan Hu, Changwei Hu, Kevin Yen, Fei Tan, Kyumin Lee and Se Rim Park, “HABERTOR: An Efficient and Effective Deep Hatespeech Detector”, to appear in Proceedings of The 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020.

11. Thanh Tran, Di You, and Kyumin Lee, “Quaternion-Based Self-Attentive Long Short-Term User Preference Encoding for Recommendation”, in Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2020.

10. Thanh Tran, Renee Sweeney, and Kyumin Lee. “Adversarial Mahalanobis Distance-based Attentive Song Recommender for Automatic Playlist Continuation”, In Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2019.

9. Thanh Tran, Xinyue Liu, Kyumin Lee, and Xiangnan Kong, “Signed Distance-based Deep Memory Recommender”, In Proceedings of the Web Conference 2019 (WWW), 2019.

8. Thanh Tran, Kyumin Lee, Yiming Liao, and Dongwon Lee, “Regularizing matrix factorization with user and item embeddings for Recommendation”, In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2018.

7. Thanh Tran, Kyumin Lee, Nguyen Vo, and Hongkyu Choi, “Identifying On-time Reward Delivery Projects with Estimating Delivery Duration in a Crowdfunding Platform”, In Proceedings of the International Conference on Advances in Social Network Analysis and Mining (ASONAM), 2017.

6. Thanh Tran and Kyumin Lee, “Characteristics of On-time and Late Reward Delivery Projects”, In Proceedings of the International AAAI Conference on Web and Social Media (ICSWM), 2017.

5. Yiming Liao, **Thanh Tran**, Dongwon Lee and Kyumin Lee, “Understanding Backing Patterns in Online Crowdfunding Communities”, In Proceedings of the International ACM Web Science Conference (WebSci), 2017.

4. Nguyen Vo, Kyumin Lee, **Thanh Tran**, “MRAttractor: Detecting Communities from LargeScale Graphs”, In Proceedings of the IEEE International Conference on Big Data (IEEE BigData), 2017.

3. Nguyen Vo, Kyumin Lee, Cheng Cao, **Thanh Tran** and Hongkyu Choi, “Revealing and Detecting Malicious Retweeter Groups”, In Proceedings of the International Conference on Advances in Social Network Analysis and Mining (ASONAM), 2017.

2. Thanh Tran and Kyumin Lee, “Understanding Citizen Reactions and Ebola-Related Information Propagation on Social Media’, In Proceedings of the International Conference on Advances in Social Network Analysis and Mining (ASONAM), 2016.

1. Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, **Thanh Tran** and Jiasheng Zhang, “Uncovering Fake Likers in Online Social Networks”, In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2016.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Many online systems such as e-commerce, music/video platforms have been proliferating in the recent decade, enabling hundreds of millions of people to experience online services. For instance, Amazon, as an e-commerce service provider, has more than 150 millions of prime consumers with more than 2.5 millions of sellers, who are actively selling on the marketplace[1], and more than 350 millions products both from Amazon and Amazon marketplace sellers. eBay, another e-commerce service provider, has 182 millions of users around the world with 1.3 billions of listings. In such online systems, there are three main objects as shown in Figure 1.1: (1) the service provider, (2) the product owners (i.e. sellers in Amazon, eBay), and (3) the consumers. The service providers provide online services for both product owners and customers so that they can interact with each other by selling, purchasing or consuming the products/services. When the online platforms present products to the customers, they may recommend relevant products to the customers to help satisfy their needs and improve customer satisfaction.

[1]https://www.oberlo.com/blog/amazon-statistics

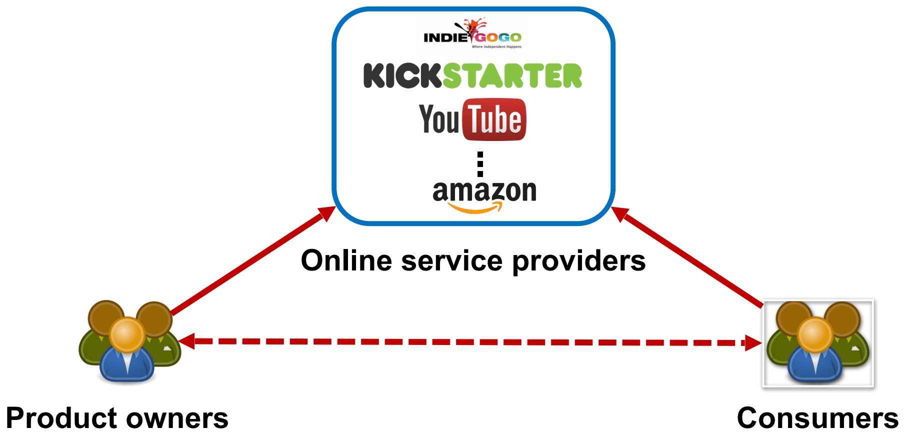


Figure 1.1: Overview of relationships among consumers, product owners, and the online service providers.

In one side, understanding and characterizing activities of product owners inherent in goods delivery have played an essential role in the prosperity of online service providers. Particularly, if the product owners deliver ordered products to consumers in a qualitative manner, i.e., deliver the products on time and in good quality, consumers will be likely to re-purchase the products in the future. This helps to maintain not only the trust between the consumers and the product owners but also the trust between the consumers/product owners and the platform providers.

In another side, studying and modeling consumer behaviors (i.e., activities associated with purchasing and consuming commodities) have been a major area in economics and marketing. From classical microeconomics to behavioral economics, researchers study consumer behaviors via various approaches such as surveys, interviews, and statistical tests. Recently, advanced technologies have supported e-commerce systems for storing massive consumer-product interaction activities, allowing researchers to step into studying consumer behaviors using the stored data, and opening possibilities of modeling consumer behaviors with personalized recommendation systems.

1.2 Research Challenges

In the previous section, we described two user groups (i.e., product owners and consumers) in online systems and the need for modeling their behaviors to prosper the success of online service providers and improve user satisfaction. In this section, we present two main research challenges when modeling user behaviors as follows:

- **High diversification and variation:** consumer's preferences are highly diverse and various from applications to applications. For example, in many problems like automatic playlist continuation in music/video streaming services (e.g. Spotify, Youtube), or shopping basket-based recommendation in e-commerce platforms (e.g. Tafeng, Tmall), *etc.*, the next product/item to be added in a target group (the playlist in Spotify/Youtube, or the basket in Tafeng/Tmall) is often decided based on the general taste of the target group. Thus, encoding the global theme of the target group based on already-added products is necessary to model the consumer's activities. However, in other applications like next movie recommendation in Netflix, next product recommendation in Amazon, *etc.*, modeling long-term dependencies of all previously consumed items is not enough. This is because the next consumed item of the consumer has a strong correlation with the consumer's current interest, which is reflected by the recently preferred items. As a result, representing the consumer's taste using the latest consumed items should not be neglected.

- **Representation Methodology:** Due to the high diversification and variations of consumer's intents, designing which representation spaces, methodologies, or modeling operators is still in demand. Specifically, Matrix Factorization-based methods have greatly succeeded in connecting consumers to relevant products, but are suboptimal in modeling complex consumer-product relationships due to a linear nature. This urges the need for designing neural network-based recommenders to encode non-linear consumer-product relationships. In another side, the dot product is mainly adopted in neural recommenders, but is still limited as it does not convey the crucial triangle inequality. As a result, given that a consumer u_1 prefers two products *iPhone 8* (i.e. item i_1) and *iPhone 8 Plus* (i.e. item i_2). Here, the two items i_1 and i_2 are similar. Learning with dot product can lead to the following two dimensional results: u=(1, 1), i_1=(1, 0), and i_2 = (0, 1), as $u_1^T i_1$ = 1, and $u_1^T i_2$ = 1. Unfortunately, $i_1^T i_2$=0, indicating the two items i_1 (iPhone 8) and i_2 (iPhone Plus) are not similar. Thus, by learning with the dot product operator, similar products/-consumers can not be depicted correctly in the high dimensional space.

1.3 Overview of this Dissertation

In this dissertation, we aim to incorporate *high diversification and variation* of user's interests into proposed models, and propose better *representation methodology* for improving online service quality and user satisfaction with making scientific innovation. Therefore, we explore the product delivery behavior of the product owners – one of the most important parts in establishing/ensuring the relationships between the consumers and the product owners, as well as the relationships between these users and the online service providers. Secondly, we aim to model the consumer preferences via two different methodological traces: (i) long-term consumer interest constraints, and (ii) long-term and short-term consumer intent constraints. To sum up, this dissertation makes the following contributions in this direction:

- **The first contribution** of this dissertation is to examine the product delivery behavior of the product owners. We study various factors affecting the product delivery phase.

  In particular, we focus on the delivery phase of promised rewards in crowdfunding activities. Note that, in crowdfunding platforms, *rewards* are considered as products, *project creators* are considered as product owners, and *investors* are considered as consumers. Also, our methods can be applied to other similar platforms. In particular, we characterize which factors made the reward delivery late/on time. Then, we extract novel features that reveal latent difficulty levels of the rewards. As a result, we build predictive models to identify whether a creator will deliver all rewards in a project on time or not. Moreover, we build a regression model to estimate accurate reward delivery duration (i.e., how long it will take to produce and deliver all the rewards) to assist the project creators in the platforms.

- **The second contribution** of this dissertation is to model consumer interests via global constraints. We consider several factors while designing a personalized recommender system: (1) which items a user likes, (2) which two users co-like the same items, (3) which two items users often co-liked, and/or (4) which two items users often co-disliked.

  Hence, we propose a joint Regularized Multi-Embedding recommendation model, which combines the weighted matrix factorization, the co-liked item embeddings, the co-disliked item embeddings, and the user embeddings, for both explicit and implicit feedback datasets. We also design a user-oriented EM-like algorithm to draw negative samples (i.e., disliked items) from implicit feedback dataset.

  Due to the fact that matrix factorization based methods have a linear representation nature, we further propose three metric learning-based neural approaches to encode non-linear relationships of consumers and items. Our approaches exploit a metric-based attention mechanism to account for similarities between consumed items and the next preferred item of a consumer, as well as utilize a Mahalanobis metric learning to constraint the distances of similar consumers/items to be closer in high dimensional space. Moreover, we improve the robustness of our models by applying adversarial personalized ranking and customizing it with a flexible noise magnitude. As a result, the proposed neural methods overcome the limitation of our RME model.

- **The third contribution** of this dissertation is to model the consumer interests via both global and local constraints. We propose models that reflect both (i) *long-term dependencies* and (ii) *short-term correlations* of the consumer's preferences.

  We design and propose a deep learning framework called *Signed Distance-based deep Memory Recommender (SDMR)*, which captures non-linear relationships between users and items explicitly and implicitly. SDMR consists of two main components: (i) a *Signed Distance-based Perceptron (SDP)* component, and (ii) a *Signed Distance-based Memory network (SDM)* component. The *SDP* module measures the signed distance between a target consumer and a target item, encoding a global interest of the target consumer. The *SDM* component measures a signed distance score between the target consumer and the target item via attentive distances between the consumer's recently consumed items and the target item, thus reflecting the consumer's short-term interest.

  While most existing recommender systems rely on the Euclidean space to represent consumers/items embeddings, we move further forward to utilize a Quaternion space to encode consumers and items latent factors. Concretely, we use Quaternion representations for all users, items and neural transformations in our proposed models. There are numerous benefits of the Quaternion utilization over the traditional real-valued representations in Euclidean space. First, Quaternion numbers/vectors consist of a real component and three imaginary components, encouraging a richer extent of expressiveness. Second, instead of using dot product in Euclidean space, Quaternion numbers/vectors operate on Hamilton product, which matches across multiple (inter-latent) Quaternion components and strengthens their inter-latent interactions, leading to a higher expressive model. Third, the weight sharing nature of the Hamilton product leads to a model with a smaller number of parameters. As a result, we propose novel Quaternion based models to learn a user's long-term and short-term interests more effectively. As a part of our framework, we propose Quaternion self-attention that works in Quaternion space. We also propose a Quaternion-based Adversarial attack on BPR-loss to improve the robustness of our models further.

1.4 Dissertation Organization

In this section, we describe the organization of this dissertation as follows:

- **Chapter 2:** In this chapter, we discuss related work about studying the delivery behavior phases in crowdfunding platforms, as well as work on the recommendation models.
- **Chapter 3:** In this chapter, we present characteristics of project creator in the reward delivery phase in crowdfunding platforms. Specifically, we present various feature types to distinguish on-time and late reward delivery projects, which further help us to investigate techniques and develop tools for late/on-time reward delivery identification of crowdfunding projects, as well as for the reward delivery duration estimation.

  The content of this chapter is extracted from the following publications:

  - **Thanh Tran** and Kyumin Lee. Characteristics of on-time and late reward delivery projects, In *Proc. of ICWSM 2017*, pages 676–679.

  - **Thanh Tran**, Kyumin Lee, Nguyen Vo, and Hongkyu Choi. Identifying on-time reward delivery projects with estimating delivery duration on kickstarter. In *Proc. of ASONAM 2017*, pages 250–257.

- **Chapter 4:** In this chapter, we present details about the two proposed models: (i) Regularized Multi-Embedding recommenders, and (ii) the three metric learning-based neural recommenders including the Mahalanobis Distance-based Recommender, the Mahalanobis distance-based Attentive Item Similarity Recommender, and the fusion between these two neural models. Both of the proposed models encode global consumer's interests using the consumer-product interaction data.

  The content of this chapter is extracted from the following publications:

  - **Thanh Tran**, Kyumin Lee, Yiming Liao, and Dongwon Lee. Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. In *Proc. of CIKM 2018*, pages 687–696.
  - **Thanh Tran**, Renee Sweeney, and Kyumin Lee. Adversarial Mahalanobis Distance-based Attentive Song Recommender for Automatic Playlist Continuation, In *Proc. of SIGIR 2019*, pages 245–254.

- **Chapter 5:** In this chapter, we focus on modeling both long-term and short-term dependencies of the consumer's preferences. Particularly, we describe our two proposals: (i) The Signed Distance-based Deep Memory Recommender, and (ii) the Quaternion-Based Self-Attentive Long Short-Term User Preference Encoding recommender.

  The content of this chapter is extracted from the following publications:

  - **Thanh Tran**, Xinyue Liu, Kyumin Lee, and Xiangnan Kong. Signed Distance-based Deep Memory Recommender. In *Proc. of WWW 2019*, pages 1841–

1852.

– **Thanh Tran**, Di You, and Kyumin Lee. Quaternion-Based Self-Attentive Long Short-Term User Preference Encoding for Recommendation, In *Proc. of CIKM 2020*.

- **Chapter 6:** We conclude with a summary of the contributions of this dissertation and provide a discussion of future research extensions to the results we presented here.

2

Related Work

In this chapter, we summarize related work on crowdfunding activities and recommendation systems as follows:

2.1 Crowdfunding activities

Researchers analyzed crowdfunding platforms [1, 2]. For example, Kuppuswamy et al. [3] showed the dynamics of Kickstarter donors. Mollick et al. [4] studied the dynamics of crowdfunding and revealed that personal networks and underlying project quality were related to the crowdfunding success. Gerber et al. [5] analyzed why people created and/or backed projects in crowdfunding platforms. Xu et al. [6, 7] showed various factors to make a project successful in terms of the fundraising. Joenssen and Müllerleile [8] analyzed 42k Indiegogo projects, and discovered that scarcity management was problematic and reduced the chances of projects to successfully achieve the fundraising goal. Researchers [9, 10, 11] studied the impact of social media and social communities in raising fund. Joenssen et al. [12] showed that timing and communication were two key factors to make projects successful.

Etter et al. [13] examined 16k Kickstarter projects and proposed a model based on pledged money features, and project and backer graph features to predict the success of the projects. Greenberg et al. [14] extracted 13 features from each of 13,000 Kickstarter projects and developed classifiers to predict project success. In [6, 15], the authors used different feature traits to predict the success of projects. Li et al. [16] analyzed 18K Kickstarter projects and built logistic and log-logistic based models to predict the chance of successfully achieving goal. Solomon et al. [17] discovered that early donation played an important role in making the project successful. Mitra et al. [18] proposed text features of project pages for the project success prediction.

Other researchers studied building recommender systems for creators and backers/investors. An et al. [19] analyzed backers' pledging behavior, and built a SVM classifier to suggest potential backers to creators. In [20], the authors used temporal, personal, geolocation and network traits to recommend a set of potential backers to projects. Rakesh et al. [21] examined a project status, personal preference of individual investors and preference of investor groups. Then, they proposed a probabilistic recommendation model to recommend projects to a group of investors.

Recently, Kim et al. [22] interviewed crowdfunding participants and found various factors which influenced backers' trust. They also conducted analysis for 4,089 delayed projects to understand how 8 factors were related to delayed duration. However, they did not study which project will pass estimated delivery date, nor how long a reward delivery including production will take.

2.2 Recommendation Systems

General Recommendation: Matrix Factorization is the most popular method to encode global user representations by using unordered user-item interactions [23, 24, 25]. Its basic idea is to represent users and items by latent factors and use dot product to learn the user-item affinity. Despite their success, they cannot model non-linear user-item relationships due to the linear nature of dot product. To overcome the limitation, neural network based recommenders were recently introduced [26, 27, 28, 29]. [26] combined a *generalized matrix factorization* component and a non-linear user-item interactions via a MLP architecture. [30, 31, 32] substituted the MLP architecture with the auto-encoder design. [33, 34] used memory augmentation to learn different user-item latent relationship. When non-existed users come with some observed interactions (i.e., recently created user accounts with some item interactions), the recommenders need to be rebuilt to generate their representations. To avoid these issues, current works encode users by combining the users' consumed item embeddings in two main streams: (i) taking average of the consumed items' latent representations [35, 36], or (ii) attentively summing [37] the consumed items' embeddings.

Sequential Recommenders: Sequential recommendation is known for its superiority to capture temporal dependencies between historical items [38]. Early works relied on Markov Chains to capture item-item sequential patterns [39, 40, 41]. Other works exploited the convolution architecture to capture more complex temporal dependencies [42]. These methods used short-term item dependencies to model a user's dynamic interest. Other sequential recommenders focused on modeling long-term user preferences using RNN-based architectures [43, 44, 45, 46]. Recent works combined both long and short-term user preferences in real-valued representations to obtain satisfactory results [47, 48, 49].

Recommendation with auxiliary information: Another line of recommender systems is to build recommendation models that take into account auxiliary information. Some deep learning based works [32, 50, 51, 52] employ auxiliary information such as item description [53], music content [54], item visual features [55, 56], reviews [57] to address the cold-start problem. However, this auxiliary information is not always available, and it limits their applicability in many real-world systems.

3

Modeling product creator behavior in product delivery activities

3.1 Introduction

Crowdfunding platforms have successfully connected millions of individual investors to creators, and helped creators to bring their ideas into the reality. In recent years, a market size of crowdfunding platforms has increased exponentially, reaching tens of billions of dollars. Among various types of crowdfunding platforms, reward-based crowdfunding platforms have become popular, especially, Kickstarter has become the most popular crowdfunding platform. According to Kickstarter[1], more than 2.5 billion dollars were pledged by approximately 12 million backers to more than 110k projects.

As shown in Figure 3.1, a project in reward-based crowdfunding platforms has two phases: (1) *the fundraising phase* – when a creator raises money by promoting the project after launching it; and (2) *the reward delivery phase* – when the creator makes and ships products as the rewards if the project was successful in terms of pledged money $\geq$ goal.

[1]https://www.kickstarter.com/help/stats

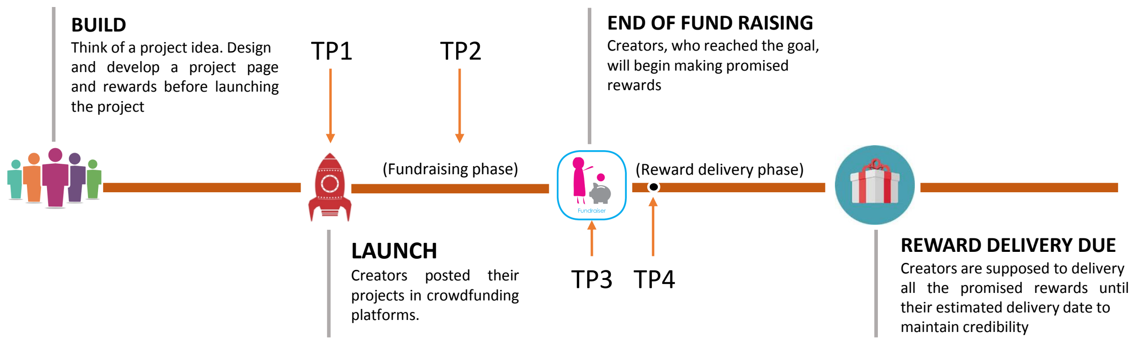

BUILD
Think of a project idea. Design and develop a project page and rewards before launching the project
TP1
TP2
(Fundraising phase)
END OF FUND RAISING
Creators, who reached the goal, will begin making promised rewards
(Reward delivery phase)
LAUNCH
Creators posted their projects in crowdfunding platforms.
TP3
TP4
REWARD DELIVERY DUE
Creators are supposed to delivery all the promised rewards until their estimated delivery date to maintain credibility

Figure 3.1: Project Timeline. In this study, we build our models at four time points (TP1, TP2, TP3 and TP4).

In the literature, researchers mostly focused on *the fundraising phase*, by analyzing dynamics of crowdfunding platforms [3], understanding why people created projects or backed other projects [5], studying how to make a project successful [6, 7, 9, 10, 11], predicting project success [13, 14, 15, 16], and recommending creators to backers or vice versa [19, 21]. However, researchers rarely paid attention to *the reward delivery phase*.

According to Kickstarter[1], 35% backers did not receive rewards on time. If creators send rewards to backers on time, backers will be likely to invest in their upcoming projects [58]. Although the time already passed the fundraising phase, if creators announce production and delivery delay with a new estimated date as soon as possible, some backers will still wait for receiving the rewards without losing much trust and without much surprise. Some backers may request a refund to creators without waiting until the estimated date (e.g., 1 year).

While on-time reward delivery becomes crucial for retaining backers in the creators' future projects, it is difficult for creators to estimate an accurate delivery date because of various reasons. First, 90% creators created a project the first time, so they don't have much experience in accurately estimating delivery date [6]. Second, some creators choose a delivery date with their hunch without understanding the reward's difficulty level. Third, there may be other uncertainties like factory issues and unexpected problems in their

[1]https://www.kickstarter.com/fulfillment

prototypes, requiring more time.

Unfortunately, little is known about what factors influence to on-time or late reward delivery projects, and there is no prior work to estimate reward delivery duration. To fill the gap, in this chapter, we focus on answering following research questions: Can we build a predictive model which can predict whether a project will be an on-time delivery project or not? Can we build a model which can estimate delivery duration accurately? Completing these two tasks would be a big challenge with only using observable online data available in a crowdfunding platform.

To answer the research questions, first we defined four time points *TP1*, *TP2*, *TP3* and *TP4*, which are when a project is launched, the middle of the fundraising phase, the end of the fundraising phase, and the first 5% of the estimated longest reward delivery duration, respectively as shown in Figure 3.1. An ideal model is supposed to predict on-time or delay at *TP1* and *TP2* well so that the investors can decide whether they are going to back the project or not. However, in practice, it would be very difficult because of many uncertainties and limited data. Therefore, building models at *TP3* and *TP4* are important and valuable as long as it can achieve high accuracy because creators can announce their delay or re-estimate reward delivery date at *TP3* and *TP4*. In addition, backers can request issuing refund at *TP3* and *TP4*. According to the refund policy of Kickstarter, it is possible for creators to refund anytime, and backers can request a refund during *the reward delivery phase*.

3.2 Dataset

This section presents our dataset with two types of ground-truth: (1) on-time or late reward delivery project and (2) actual delivery duration.

Ground truth collection: We define an on-time reward delivery project and a late reward delivery project as follows:

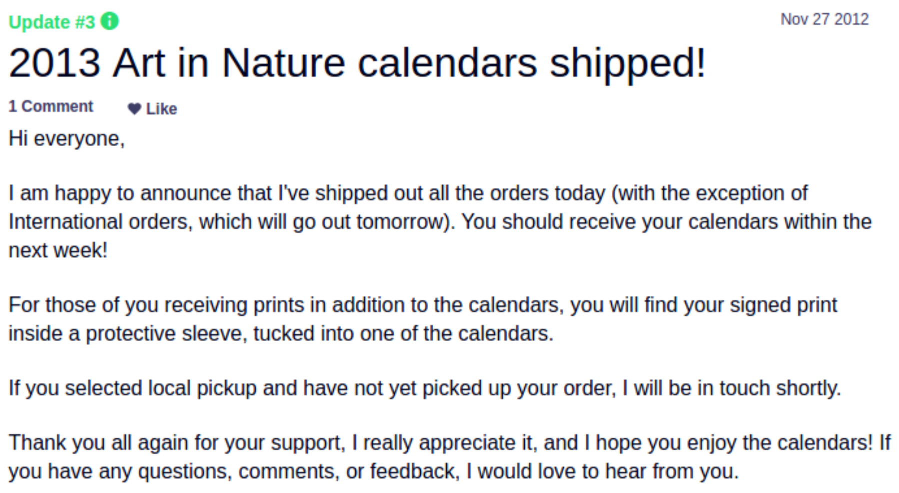
- On-time reward delivery project: If all rewards in a project were shipped by the longest estimated delivery date (LEDD), it would be called an on-time reward delivery project. Note that a project creator decides each reward's estimated delivery date when she creates her project page.
- Late reward delivery project: If a creator did not ship at least one of rewards by the LEDD, the project would be called a late reward delivery project.

We collected all project pages (i.e., 168,851 project pages) from Kickstarter which were created between 2009 and September 2014. Among the 168,851 projects, we extracted successful projects, each of which had a project goal equal to or greater than \$100. 29,499 successful projects satisfied the condition. In addition, we collected updates and comments associated with the successful projects.

Labeling each project for delivery status and duration requires reading all the updates and comments. Instead of labeling all the successful projects, we sampled 10% of the 29,499 successful projects with keeping the same project distribution over project categories, year and goal. Then, three labelers independently labeled the 2,949 sampled projects based on the following guideline:

- If a labeler could identify that all rewards in a project were shipped by LEDD (based on updates and comments), and there was no complaint regarding not receiving the rewards, she would label the project as an on-time reward delivery project.
- If there was at least one update from a creator after LEDD regarding delayed shipping or a comment with a new delivery date beyond LEDD, a labeler labeled the project as a late reward delivery project.

We excluded projects if labelers were not able to verify whether a project is an on-time reward delivery project or not based on the labeling guideline. Finally, 2,198 projects were


Update #3

Nov 27 2012

2013 Art in Nature calendars shipped!

1 Comment ♥ Like

Hi everyone,

I am happy to announce that I've shipped out all the orders today (with the exception of International orders, which will go out tomorrow). You should receive your calendars within the next week!

For those of you receiving prints in addition to the calendars, you will find your signed print inside a protective sleeve, tucked into one of the calendars.

If you selected local pickup and have not yet picked up your order, I will be in touch shortly.

Thank you all again for your support, I really appreciate it, and I hope you enjoy the calendars! If you have any questions, comments, or feedback, I would love to hear from you.

Figure 3.2: An update containing shipping information.

labeled by them, and consisted of 1,003 on-time and 1,195 late reward delivery projects.

Next, we were interested in collecting true/actual delivery duration as the ground truth (i.e., how long it took to deliver all the rewards since the end of the fundraising phase). Out of 2,198 projects, the creators of 1,598 projects posted updates with information when they shipped all the rewards, as shown in Figure 3.2. Based on the information, true/actual delivery duration of the 1,598 projects was calculated.

Categorical distribution: Figure 3.3 shows the categorical distributions of on-time and late reward delivery projects. There was a higher probability of on-time reward delivery in dance and theater-related projects because the rewards in those categories were often live performance, show cases or dancing tutor classes, and were served at once for all backers. In contrast, the rewards in other categories like games, technology, film were real products (e.g., a game, book, movie), requiring more time to produce and deliver to backers.

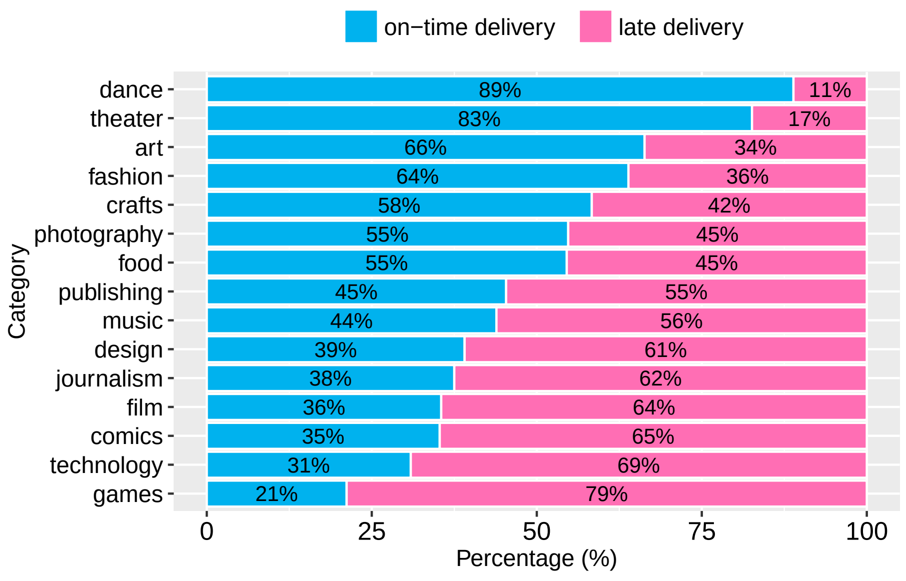


Figure 3.3: Category distributions of on-time and late reward delivery projects.

3.3 Feature Engineering

3.3.1 Latent Reward Difficulty Features

In this section, we propose a novel approach to measure a reward's difficulty level and a project's overall difficulty level toward extracting features, which will be a part of our final feature set for building models in the following sections. Our hypothesis is that true delivery duration for a reward depends on its difficulty level, and reward description may reveal the difficulty level. It makes sense that developing a game as a reward requires more time and effort than producing a t-shirt. In this section, we study how to measure the difficulty level of each reward and represent how hard a project is in terms of producing and delivering its rewards.

Clustering approach to get new features: We group rewards into the same cluster if their descriptions contain semantically similar meaning. Intuitively, if two reward descriptions are semantically similar, they may have similar difficulty level and thus require a similar amount of time to produce and deliver.

Our approach consists of six steps as follows:

- Step 1: 1,273,617 rewards were extracted from 149,189 Kickstarter projects, and their reward descriptions were preprocessed by removing stop words and punctuation.

- Step 2: Using 1,273,617 reward descriptions, we built Glove model [59], in which each word is represented by a vector. In our implementation, we set up vector size=50, maximum number of iterations=20, window size=15, and vocabulary minimum count=5.

- Step 3: From Step 2, all words in the 1,273,617 reward descriptions were represented by Glove vectors. We grouped the words into *K1* clusters by running k-means clustering algorithm. To choose the optimal *K1*, we varied *K1* from 1 to 100 and selected the value that minimized BIC value as follows:

$$BIC = \sum_{k=1}^{K1} \sum_{i \in words_k} dist(v_i, c_k) + log(n) * m * K1 \tag{3.1}$$

  where v_i, c_k is the representative vector of the word i in cluster k and the center of cluster k, respectively, $dist(v_i, c_k)$ is the Euclidean distance of two vectors v_i and c_k, n is the number of rewards (e.g. $n = 1,273,617$), m is the number of dimensions of the word vector, and $K1$ is the number of clusters. Finally, the optimal K1 was 67. So at step 3, we clustered words into 67 groups.

- Step 4: From the sampled and labeled 2,198 projects, we extracted 19,266 reward descriptions.

- Step 5: We represented each of 19,266 reward descriptions to a vector with 67 dimensions, each of which is mapped with a word cluster in Step 3. In particular, we counted how many words in each cluster occurred in the reward description, and used the count as a value of the dimension mapped to the cluster.

- Step 6: Each of 19,266 rewards/reward descriptions was represented by a vector in 67 dimensions. Then, we clustered the rewards into 14 groups. Like Step 3, we did the same process finding the optimal number of clusters. We call the 14 groups as 14 *semantic reward clusters*.

By doing the six steps, we got 14 semantic reward clusters. Then, we generated 14 feature values for each of 2,198 projects as follows: given a project p_k, rewards in p_k and corresponding number of backers to each reward, we summed up the number of backers of each reward that belongs to ith semantic cluster c_i, and used it as a feature value. We considered both a difficulty level of each reward and the corresponding number of backers because more backers mean the creator has to produce more number of outcomes. Finally, the project p_k was represented by a vector in 14 dimensions.

Quality of the clusters: To prove that each semantic reward cluster has a distinguishing difficulty level, first we identified each project p_k's major semantic cluster $M(p_k)$, a cluster (i.e., one of the 14 clusters) to which the large number of rewards in p_k belongs. Then, given a set of on-time delivery projects A, we defined an indicator function $\mathbf{1}_A$ of the project p_k as follows:

$$\mathbf{1}_A(p_k) = \begin{cases} 1 & \text{if } p_k \in A \\ 0, & \text{if } p_k \notin A \end{cases}$$

Then, given a probability of each semantic reward cluster $P(c_i)$, we calculated the conditional probability of the project p_k to be an on-time delivery project using Bayes theorem as follows:

$$P(\mathbf{1}_A(p_k) = 1 | c_i = M(p_k)) = \frac{P(c_i = M(p_k) | \mathbf{1}_A(p_k) = 1) * P(\mathbf{1}_A(p_k) = 1)}{P(c_i = M(p_k))}$$

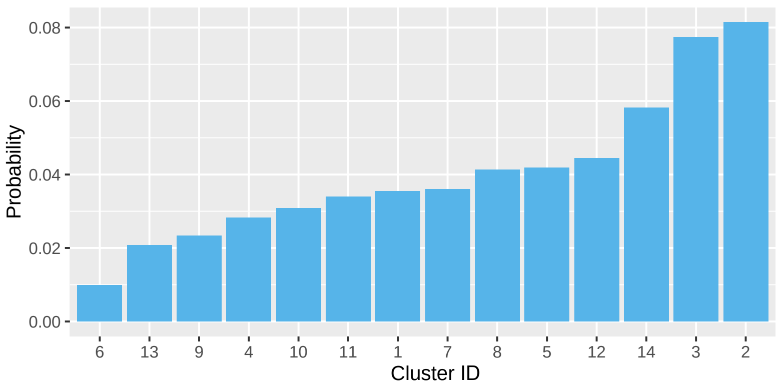
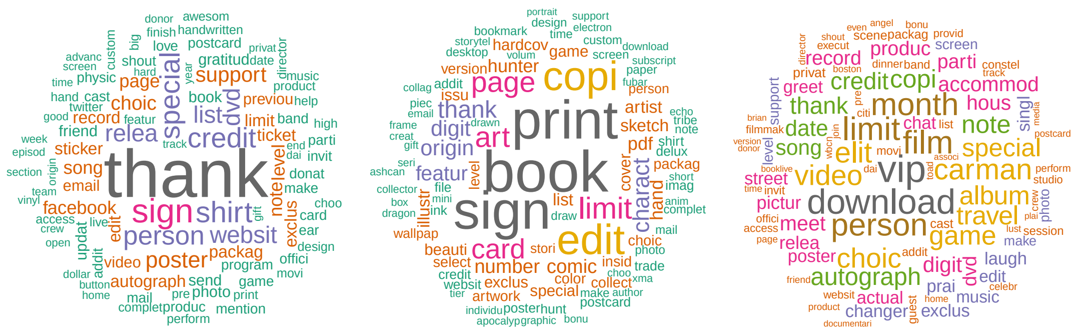


Figure 3.4: 14 semantic reward clusters with their on-time delivery conditional probabilities in the descending order of difficulty levels.



(a) Easy level (b) Medium level (c) Hard level

Figure 3.5: Word clouds of three clusters: cluster 2 (easy level), cluster 1 (medium level) and cluster 6 (hard level).

where $P(c_i = M(p_k))$ is P(p_k's major semantic cluster).

Figure 3.4 presents the conditional probability of $P(\mathbf{1}_A(p_k) = 1|c_i = M(p_k))$ in each semantic reward cluster by descending order of difficulty level. Each cluster had a different probability. Semantic reward cluster *6* had the lowest probability, indicating that rewards in this group had a higher difficulty level than other groups (e.g. hard level). In contrast, semantic reward cluster *2* had the highest probability, showing that the rewards in this cluster were easier in terms of producing and shipping (e.g. easy level). Semantic reward cluster *1* had a middle probability (e.g. medium level). We next plotted the word clouds of those three clusters to understand what kind of rewards were included in

Table 3.1: Features that were newly extracted at each time or phase.

At the launching time
Project based features: |images|, |faqs|, goal, project category, |rewards|, |reward sentences|, |bio description sentence|, fund raising duration, the longest reward delivery duration, SMOG score of project, reward and bio description, and semantic reward clustering features.
During the fundraising phase
Project based features: |backers|, |project's comments|, |project's updates|. *Creator's activeness features*: |creator's comments|, |creator's updates|. *Temporal features*: |comments| in each of 20 time slots.
At the first 5% of the longest reward delivery duration
Creator's activeness features: |creator's comments|, |creator's updates|, average update time interval, and average response time between a backer's question and a reply from the creator. *Backer's activeness features* : |backers' comments|, |backers who posted comments|, |backers' questions|. *Linguistic features:* LIWC feature extracted from updates (12 scores), LIWC feature extracted from comments (5 scores).

the three clusters. In Figure 3.5, we observed that the rewards in easy level (cluster 2) mostly contained keywords like "thank, credit, shirt, sign, websit" which can be delivered quickly. Rewards in the medium level (cluster 1) contained keywords like "print, book, sign, copi", related to publishing category. Rewards in the hard level (cluster 6) contained keywords like "film, vip, video, game", related to the film and game category. As shown in Figure 3.3, it makes sense that film, games were those categories with the highest late delivery rate, whereas publishing category had a lower late delivery rate. In the following section, we show that adding semantic cluster features improved our prediction rate.

Table 3.2: Top 10 features at TP1, TP2, TP3 and TP4

Launching time (TP1)		Middle of fundraising (TP2)		End of fundraising (TP3)		First 5% of reward delivery duration (TP4)	
Features	z-score	Features	z-score	Features	z-score	Features	z-score
project category	24.11	project category	22.93	|updates|	22.47	|creator's updates| (at TP4)	35.73
goal	18.63	goal	15.24	project category	17.47	|creator's comments| (at TP4)	18.19
longest reward delivery duration	13.40	longest reward delivery duration	10.99	goal	12.76	project category	14.80
smog score of project's description	9.71	|temporal Comment at slot 1^{st}|	8.52	|creator's comments|	11.70	longest reward delivery duration	13.60
semantic reward cluster 9^{th}	7.26	|temporal Comment at slot 8^{th}|	8.38	longest reward delivery duration	10.23	|backer's comments| (at TP4)	13.16
semantic reward cluster 2^{nd}	6.67	|temporal Comment at slot 9^{th}|	8.29	|temporal Comment at slot 18^{th}|	7.65	|project's comments|	12.69
|images|	5.90	|temporal Comment at slot 2^{nd}|	7.94	|temporal Comment at slot 19^{th}|	7.43	average update time interval	12.07
|Faqs|	4.97	|temporal Comment at slot 7^{th}|	7.74	|temporal Comment at slot 17^{th}|	7.08	goal	10.76
semantic reward cluster 12^{th}	4.89	|temporal Comment at slot 4^{th}|	7.73	|temporal Comment at slot 14^{th}|	6.92	|creator's comment| (at TP3)	8.16
semantic reward cluster 7^{th}	4.45	smog score of project's description	7.67	|temporal Comment at slot 16^{th}|	6.88	|temporal comment at slot 18^{th}|	6.53

3.3.2 Other Feature Sets

Toward identifying on-time and late reward delivery projects as well as estimating reward delivery duration for the projects, we extracted the following features and used in the following sections:

- Project-based features: We extracted 16 project-related features: |images|, |faqs|, goal, project category, |rewards|, |reward sentences|, |bio description sentence|, fund raising duration, the longest reward delivery duration, SMOG score [60] of project description, SMOG score of reward description, SMOG score of bio description, |backers|, |project's comments|, |project's updates|, and semantic reward clustering feature.

- Creator activeness features: We extracted 4 features related to the creator's activeness: |creator's comments|, |creator's updates|, average update time interval, and average response time between a backer's question and a reply from the creator.

- Backer's activeness features: Backers can post comments and ask for the project progress. We extracted 3 features related to the backer's activeness: |backers' comments|, |backers who posted comments|, |backers' questions|.

- Temporal features: We converted each project's fundraising duration into 20 states (time slots), since projects have various fundraising periods (e.g., 30 days, 60 days).

In each state/time slot, we measured the number of comments posted by creators and backers.

- Linguistic usage of creators and backers: We used Linguistic Inquire and Word Count (LIWC) dictionary [61] to discover distinguished linguistic usage patterns of creators (through their updates). To compute the linguistic usage score of creators in on-time reward delivery projects and creators in late reward delivery projects over 64 LIWC categories, we performed the same process which was mentioned in [62, 63]. We applied two-sample t-test and assigned α as 0.00078 (=0.05/64) to select only the LIWC categories in which we observed a significant difference between two distributions. Finally, we found 12 LIWC categories in which creators in on-time and late reward delivery projects had a significant linguistic-usage difference. Via the same process, we measured the different linguistic usage of backers by their comments. We found 5 LIWC categories in which backers in on-time and late reward delivery projects had a significant linguistic-usage difference.

3.4 Identifying On-time and Late Reward Delivery Projects

In this section, we build predictive models to classify whether a project is an on-time delivery project or not and evaluate their performance against baselines.

3.4.1 Experimental Setting

As shown in Figure 3.1, we conducted experiments at the four time points, depending on what information available at each time point: (1) **TP1**: when a project is launched; (2) **TP2**: in the middle of the fundraising phase; (3) **TP3**: in the end of the fundraising phase; and (4) **TP4**; at the first 5% of the longest reward delivery duration (to see whether building classifiers in 5% delivery duration improve the classification performance compared with TP1, TP2 and TP3). At each time point, we extracted available features and conducted 10-fold cross-validation. Table 3.1 presents our proposed features. We added previously available features to following time points.

At each time point, we did feature selection by removing linearly related features and unimportant features. Particularly, to remove linearly related features, we measured the *variance inflation factor* (VIF) of each feature. VIF value of a feature i is computed as follows:

$$VIF_i = \frac{1}{1 - R_i^2}$$

where R_i^2 is the coefficient of multiple determination obtained by doing regression of the feature i as response to the remaining features. A VIF value is in a range of $[0, \infty)$. If a feature i's VIF value is 1, it means there is no correlation between the feature and the other features. But, if its VIF value is equal to 10 or greater, it means there exists multicollinearity. Removing the multicollinearity follows three steps: *Step 1*: VIF values of all *n* features are computed; *Step 2*: If some features have VIF score$\geq$10, we remove the feature with the largest VIF value and recompute VIF values of *n-1* remaining features; *Step 3:* If all VIF values of *n-1* features are less than 10, we stop this process. Otherwise, we repeat step 2.

To remove unimportant features, we used Boruta algorithm [64]. Boruta exploited a Random Forest classification algorithm to measure feature importance score (e.g *z-score*). For each feature, Boruta produces some statistical scores (e.g. mean, max, min, median of *z-score*), and one of three statuses: *confirmed*, *tentative* or *rejected*. *Confirmed* features are important features while *rejected* features are unimportant features. *Tentative* features are those with *insured* importance. We kept only *confirmed* features in our models.

Table 3.2 shows top 10 features at each of the four time points. Project category, goal and the longest reward delivery date were in top 10 features at all the time points. Seman-

Table 3.3: Prediction results for on-time and late delivery projects. The improvement of XGBoost model over the baselines was significant with p-value < 0.001 using the Wilcoxon directional test.

Approach	Accuracy at four time points			
	TP1	TP2	TP3	TP4
Baseline 1	54.36%	54.36%	54.36%	54.36%
Baseline 2 [22]	62.65%	62.65%	63.10%	63.10%
Our Model	**65.8%**	**66.4%**	**71.4%**	**82.5%**

tic reward cluster features were in top 10 features at TP1, and also important features at the other times points but not in top 10 features. Temporal features were in top 10 features at TP2 and TP3, whereas creator and backer's activity features were in top 10 features at TP4.

Even though there is no prior work directly related to reward delivery status prediction, we implemented two baselines to compare with our approach:

- *baseline 1*: It is the majority class selection approach by blindly predicting all projects as late reward delivery projects (54.36%).
- *baseline 2:* Kim et al. [22] measured the number of delayed days by 8 features: # of rewards, goal, project duration, # of backers, percent raised, # of backed projects, # of created projects, and project type. We built XGBoost model with the 8 features as a baseline. Note that # of backers and percent raised would be available only at TP3 and TP4.

To build our predictive model, we built a XGBoost model based on all the features that we listed in Table 3.1. We also tried Naive Bayes, SVM and Random Forest based classifiers, but XGBoost classifiers achieved the best results with our features.

3.4.2 Experiment Results

Table 3.3 shows experimental results of the two baselines and our predictive model. Our XGBoost classifier achieved 65.8%, 66.4%, 71.4% and 82.5% accuracy at TP1, TP2, TP3, and TP4, respectively. It outperformed the baselines at all the time points, especially, significantly improving the accuracy by 13$\sim$31% at TP3 and TP4 against the best baseline (p-value < 0.001). The experimental results revealed that achieving high prediction rates at TP1 and TP2 were very difficult with only using limited observable online data available in Kickstarter because of other factors (e.g., factory issues and unexpected problems) even though our model was better than the baselines. However, in the bright side, our model achieved 82.5% accuracy at TP4, so that it can notify to a project creator, backers and the platform provider whether the rewards delivery will be delayed or not. The creator can announce this news and let backers know in advance (only passing the first 5% of the longest delivery duration), or the backers can request a refund if they don't want to wait for longer time period.

Next, we analyze which feature group had more distinguishing power between on-time and late delivery projects. In this study, we focus on TP4 containing all the features. In each time, we excluded one of the four feature groups: *creator's activeness*, *backer's activeness*, *linguistic* and *semantic reward cluster* features presented in Table 3.1. Then we built a XGBoost classifier based on the remaining features.

Table 3.4 presents experimental results. Removing linguistic features, semantic reward cluster features, backer's activeness features and creator's activeness features reduced the accuracy by 1.7%, 2%, 2.4%, and 10%, respectively. Overall, creator's activeness features were the most important feature group, even though the other feature groups were also important.

In summary, the experimental results confirmed that our proposed approach was significantly better than the baselines, especially achieving 82.5% accuracy at TP4.

Table 3.4: Feature analysis to understand which feature group degrades our model's performance. The accuracy difference between our model (using all the features) and the rest models are significant with p-value<0.01 using two-sample t-test.

Model	Accuracy
All	82.5%
All - *linguistic features*	80.8%
All - *semantic clustering features*	80.5%
All - *backer's activeness*	80.1%
All - *creator's activeness*	72.5%

Table 3.5: Prediction results estimating delivery duration at four time points. The improvement of our model over the baseline was significant with p-value $<$ 0.001 using the Wilcoxon directional test.

Algorithm	RMSE (days)				NRMSE@A				NRMSE@B			
	TP1	TP2	TP3	TP4	TP1	TP2	TP3	TP4	TP1	TP2	TP3	TP4
Baseline [22]	180.0	180.0	173.0	173.0	0.248	0.248	0.239	0.239	0.451	0.451	0.434	0.434
Our models:												
semantic reward cluster features	106.8	106.8	106.8	106.8	0.148	0.148	0.148	0.148	0.268	0.268	0.268	0.268
other features	106.1	106.1	105.6	93.2	0.146	0.146	0.145	0.128	0.266	0.266	0.265	0.234
All features	**100.7**	**100.4**	**94.1**	**78.1**	**0.139**	**0.138**	**0.130**	**0.108**	**0.252**	**0.252**	**0.236**	**0.196**

3.5 Predicting Rewards Delivery Duration

The previous experimental results motivated us to study how to estimate a project's longest reward delivery duration. What if we can estimate delivery duration accurately at TP1, TP2, TP3 and TP4 in an automated way, it will help the creator to decide better longest reward delivery duration at TP1 or re-estimate it at TP2, TP3 or TP4. In backers' perspective, some backers may be willing to wait longer as long as the project creator notify re-estimated delivery duration in advance (say, TP4) [22]. Therefore, in this section, we build our regression model to estimate delivery duration (in days), and then evaluate its performance compared with 1,598 projects' ground truth described in the Dataset Section.

3.5.1 Experimental Setting

In this study, we used the features presented in Table 3.1 and applied stepAIC algorithm to choose important features. Then we conducted 10-fold cross-validation for our experiment.

Evaluation Metrics: To evaluate each model's performance, we used Root Mean Squared Error (RMSE), and normalized root mean squared error (NRMSE) with regard to the range and the mean of the ground truth data. Lower RMSE and NRMSE values indicate a better model. In the literature, researchers used two versions of NRMSEs, so we also used both of them as evaluation metrics:

$$\text{NRMSE@A} = \frac{RMSE}{max(y) - min(y)}; \text{NRMSE@B} = \frac{RMSE}{\bar{y}}$$

where $max(y)$, $min(y)$ and $\bar{y}$ are the maximum, the minimum and the mean of the ground truth values, respectively.

Data Transformation: Given a project i, we denote $h(x^{(i)})$ as the estimated number of days the creator needs to deliver all promised rewards, y_i as the ground truth of the project (e.g. the actual/true number of days that the creator needed to deliver all promised rewards), and $x^{(i)}$ as the feature vector. Before building our regression model, we performed 2-step data transformation as follows:

- Transformation on feature values $x^{(i)}$: all feature values in feature vector $x^{(i)}$ of a project i were log transformed. In particular, we used $x^{(i)} = log(1 + x^{(i)})$ instead of using the original feature values.

- Transformation on ground truth y_i: we used box-cox transformation [65] to transform y_i as follows:

$$y_{new}^{(i)} = \begin{cases} \frac{y_i^{\lambda}-1}{\lambda}, & \text{if } \lambda \neq 0 \\ log(y_i), & \text{if } \lambda = 0 \end{cases}$$

To choose the best value of λ, we fit a multiple linear regression $h(x) = \sum_{i=0}^{n}(\theta_i x_i + \epsilon_i) = \theta^T x + \epsilon$. We assumed all the errors ϵ are independent and $\epsilon \sim \mathcal{N}(0, \sigma^2)$. After doing transformation, the log-likelihood $\mathcal{L}$ of the model, with regard to the value of λ, is calculated as below:

$$\mathcal{L} = -\frac{n}{2} log\left[\sum_{i=1}^{m}\left(\frac{y_{new} - h_\theta(x^{(i)})}{exp(\frac{1}{n}\sum_{i=1}^{m} log(y_i))^{\lambda}}\right)^2\right]$$

We varied the value of λ in the grid [-1, 1] and selected $\lambda = 0.11$ since it maximized the log-likelihood $\mathcal{L}$. However, if we transform $y_{new} = y^{\lambda}$, the magnitude of the sum of squared error (SSE) will be changed. In other words, the value of SSE will depend on λ. To overcome this issue, we normalized y_{new} by the geometric mean of all raw values y_i as below:

$$y_{new} = \frac{(y_i^{\lambda} - 1)}{\lambda \prod_{i=1}^{n}(\sqrt[n]{y_i})^{\lambda-1}}$$

Our Model: To predict how many days the creator of a project i needs to fully deliver all the promised rewards, we built a multiple linear regression model based on the features in Table 3.1 except the *longest reward delivery duration* feature. Let x be the feature vector ($x_0 = 1$), m be |projects|, θ be the coefficient vector of the feature vector (except that θ_0 is the intercept), and ϵ_i be the error term which follows normal distribution. We used the squared loss and introduced elastic net regularization to find the optimal values of the coefficient vector θ by minimizing the following loss function:

$$\underset{\theta}{min}\mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}(\sum_{j=1}^{n}\theta_{ij}x_{ij} - y_i)^2 + \lambda_1||\theta||_1 + \lambda_2||\theta||_2^2 \qquad (3.2)$$

Here, λ_1 and λ_2 were used to control the regularization effect.

Baseline: Kim et al. [22] proposed 8 features and built a simple multiple linear regression to measure the number of delayed days (e.g. the difference in days between estimated reward delivery date and real reward delivery date).

3.5.2 Experiment Results

Table 3.5 shows experimental results of all the methods at TP1, TP2, TP3, and TP4. We implemented 3 models with different feature sets: (i) *semantic reward cluster features*; (ii) *other features*: all features presented in Table 3.1 except semantic reward cluster features and *longest reward delivery duration* feature; and (iii) *all features*: combining (i) and (ii). All of our models outperformed the baseline. Interestingly, the *semantic reward cluster features* model achieved almost same result with the *other features* model at TP1, TP2 and TP3. It indicates that the semantic reward cluster features were helpful in estimating the reward delivery duration. Adding available features at TP4 (e.g. creator's activeness features + backer's activeness features + linguistic features) reduced 12% error of the *other features* model. The *all features* model gained the best result with lowest RMSE=100.7, NRMSE@A=0.139, and NRMSE@B=0.252 at TP1, significantly reduced 44% error compare to the baseline. It also significantly reduced RMSE, NRMSE@A and NRMSE@B by minimum 45% and 55% at TP3 and TP4, respectively compared with the baseline. Comparing with NRMSE@A of another model in another domain [66], our model performed well, indicating the effectiveness of our regression model. We also tried SVM, Neural Network and Random Forest based on our features to build regression models. But, the multiple linear regression with the elastic net regularization got the best result.

In this context, RMSE=78.1 means the average difference between our estimated delivery duration and the ground truth (i.e., actual delivery duration) is 78.1 days. In a real-world scenario, project creators at TP1 can use our model to estimate delivery duration. Then, they can add 101 days to the estimated delivery duration as a buffer to make sure they can get enough delivery duration. Similarly, creators at TP4 can get estimated delivery duration from our model, and add 78 days as a buffer to the estimated delivery duration. The final delivery duration will be safe/enough delivery duration to deliver reward within the longest delivery duration.

We further investigated to understand whether there is any significant difference between the most correctly predicted projects and the most incorrectly predicted projects. In particular, we extracted top 10 correctly predicted projects and top 10 incorrectly predicted projects. Then we conducted Wilcoxon test which showed that top 10 correctly predicted projects provided a larger number of projects and contained a larger number of sentences in their reward descriptions compared with top 10 incorrectly predicted projects.

4

Modeling consumer behaviors with long-term dependencies

4.1 Introduction

In this chapter, we investigate and come up with our model to encode the consumer behaviors with long-term interacted item dependencies, thus recommending more relevant products to consumers.

Among popular *Collaborative Filtering* (CF) methods in recommendation [23, 24, 67, 68], in recent years, latent factor models (LFM) using matrix factorization have been widely used. LFM are known to yield relatively high prediction accuracy, are language independent, and allow additional side information to be easily incorporated and decomposed together [69, 70]. However, most of conventional LFM only exploited positive feedback while neglected negative feedback and treated them as missing data [23, 71, 72, 73].

We observe three global constraints while modeling consumer's long-term dependencies. Taking movie recommender systems as an example, it was observed that many users who enjoyed watching *Thor: The Dark World*, also enjoyed *Thor: Ragnarok*. In this case, *Thor: The Dark World* and *Thor: Ragnarok* can be seen as a pair of co-liked movies. So, if a user preferred *Thor: The Dark World* but never watch *Thor: Ragnarok*, the system can precisely recommend *Thor: Ragnarok* to her (**first observation**). Similarly, if two users A and B liked the same movies, we can assume A and B have the same movie interests. If user A likes a movie that B has never watched, the system can recommend the movie to B (**second observation**). In the same manner, we ask if co-occurred disliked movies can provide any meaningful information. We observed that most users, who rated *Pledge This!* poorly (0.8/5.0 on average), also gave a low rating to *Run for Your Wife* (1.3/5.0 on average). If the disliked co-occurrence pattern was exploited, *Run for Your Wife* would not be recommended to other users who did not enjoy *Pledge This!* (**third observation**). This will help reduce the false positive rate for recommender systems. The same phenomena would have also occurred in other recommendation domains.

The first two observations are similar to the basic assumptions of item CF and user CF where similar scores between items/users are used to infer the next recommended items for users. Unfortunately, only the first two observations have been exploited in conventional CF. While treating the negative-feedback items differently from missing data led to better results [74], to the best of our knowledge, no previous works exploited the **third observation** to enhance the recommender systems' performance.

Therefore, in this chapter, we attempt to exploit all three observations in one model to achieve better recommendation results. With the recent success of word embedding techniques in natural language processing, if we consider pairs of co-occurred liked/disliked items or pairs of co-occurred users as pairs of co-occurred words, we can apply word embedding to learn latent representations of items (e.g., item embeddings) and users (e.g. user embeddings). Based on this, we propose a *Regularized Multi-Embedding* based recommendation model (RME), which jointly decomposes (1) a user-item interaction ma-

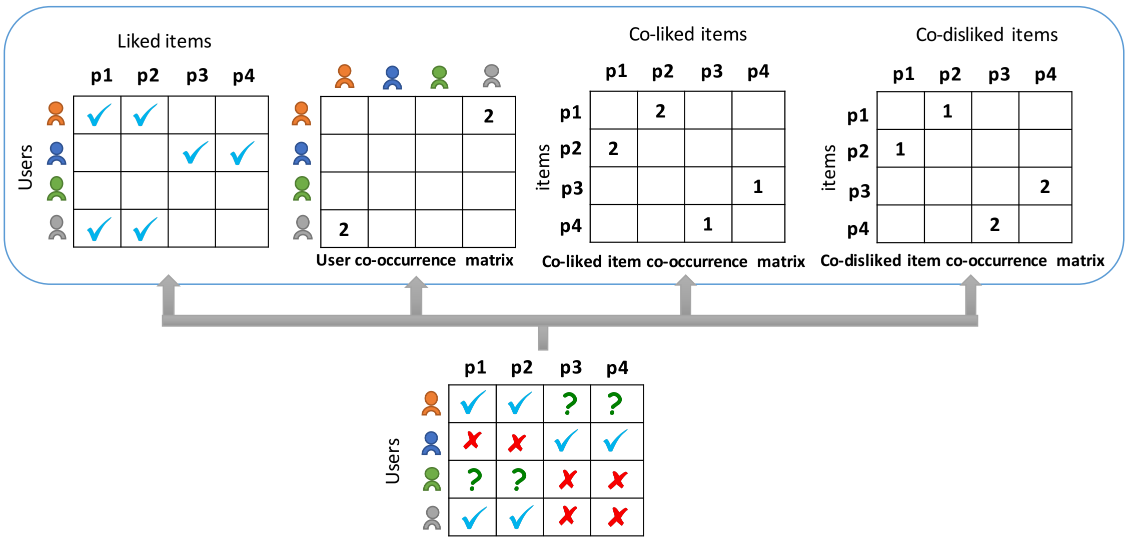

Liked items
Users
p1 p2 p3 p4
User co-occurrence matrix
Co-liked items
items
Co-liked item co-occurrence matrix
Co-disliked items
Co-disliked item co-occurrence matrix

Figure 4.1: An overview of our RME Model, which jointly decomposes user-item interaction matrix, co-liked item co-occurrence matrix, co-disliked item co-occurrence matrix, and user co-occurrence matrix. (V: liked, X: disliked, and ?: unknown)

trix, (2) a user co-occurrence matrix, (3) a co-liked item co-occurrence matrix, and (4) a co-disliked item co-occurrence matrix. The RME model concurrently exploits the co-liked co-occurrence patterns and co-disliked co-occurrence patterns of items to enrich the items' latent factors. It also augments users' latent factors by incorporating user co-occurrence patterns on their preferred items. Figure 4.1 illustrates an overview of our RME model.

Both liked and disliked items can be explicitly measured by rating scores (e.g., a liked item is $\geq$ 4 star-rating and a disliked item is $\leq$ 2 star-rating) in explicit feedback datasets such as 5-star rating datasets (e.g., a Movie dataset and an Amazon dataset). However, in implicit feedback datasets (e.g., a music listening dataset and a browsing history dataset), users do not explicitly express their preferences. In *implicit feedback datasets*, the song plays and URL clicks could indicate how much users like the items (i.e., positive samples), but inferring the disliked items (i.e., negative samples) is a big challenge due to the nature of implicit feedback. In order to deal with this challenge, we propose an algorithm which infers a user's disliked items in implicit feedback datasets, so that we can build an RME model and recommend items for both explicit and implicit feedback datasets.

4.2 Recommending Products with Regularized User and Item Embeddings

4.2.1 Method

Preliminaries: Before discussing about the model details, we define some notations and terminologies that we will use in our model description as follows:

Item. Items are objects that users interact with or consume. They can be interpreted in various ways, depending on the context of a dataset. For example, an item is a movie in a movie dataset such as MovieLens, whereas it is a song in TasteProfile.

Liked items and disliked items. In explicit feedback datasets such as MovieLens (a 5-star rating dataset), an item ≥ 4 stars is classified to a liked item of the user, and an item ≤ 2 stars is classified to a disliked item of the user [75]. In implicit feedback datasets such as TasteProfile, the more a user consumes an item, the more he/she likes it (e.g., larger play count in TasteProfile indicates stronger preference). But, disliked items are not explicitly observable.

Top-N recommendation. In this chapter, we focus on top-N recommendation scenario, in which a recommendation model suggests a list of top-N most appealing items to users. We represent the interactions between users and items by a matrix M^{m*n} where *m* is the number of users and *n* is the number of items. If a user u likes an item p, M_{up} will be set to 1. From *M*, we are interested in extracting co-occurrence patterns including liked item co-occurrences, disliked item co-occurrences, and user co-occurrences. Our goal is to exploit those co-occurrence information to learn the latent representations of users and

Table 4.1: Notations.

Notation	Description
M	a $m \times n$ user-item interaction matrix.
U	a $m \times k$ latent factor matrix of users.
P	a $n \times k$ latent factor matrix of items.
X	a $n \times n$ SPPMI matrix of liked items-item co-occurrences.
Y	a $n \times n$ SPPMI matrix of disliked item-item co-occurrences.
Z	a $m \times m$ SPPMI matrix of user-user co-occurrences.
α_u	a $k \times 1$ latent factor vector of user u.
β_p	a $k \times 1$ latent factor vector of item p.
γ_i	a $k \times 1$ latent factor vector of co-liked item context i.
$\delta_{i\prime}$	a $k \times 1$ latent factor vector of co-disliked item context $i\prime$.
θ_j	a $k \times 1$ latent factor vector of user context j.
λ	a hyperparameter of regularization terms.
b, d	co-liked and co-disliked item bias.
c, e	co-liked and co-disliked item context bias.
f, g	user bias and user context bias.
w_{up}	a weight for an interaction between user u and her liked item p.
$w_{uj}^{(u)}$	a weight for two users u and j who co-liked same items.
$w_{pi}^{(+p)}$	a weight for two items p and i that are co-liked by users.
$w_{pi}^{(-p)}$	a weight for two items p and i that are co-disliked by users.

items, then recommend top-N items to the users.

Notations. Table 4.1 shows key notations used in this chapter. Note that all vectors in this chapter are column vectors.

Next, we review the Weighted Matrix Factorization (WMF), and co-liked item embedding. Then, we propose co-disliked item embedding and user embedding. Finally, we describe our RME model and present how to compute it.

Weighted matrix factorization (WMF). WMF is a widely-used collaborative filtering method in recommender systems [23]. Given a sparse user-item matrix $M^{m\times n}$, the basic idea of WMF is to decompose M into a product of 2 low rank matrices $U^{m\times k}$ and $P^{n\times k}$ (i.e., $M = U \times P^T$), where k is the number of dimensions and $k < min(m, n)$. Here, U is interpreted as a latent factor matrix of users, and P is interpreted as a latent factor matrix of items.

We denote $U^T = (\alpha_1, \alpha_2, ..., \alpha_m)$ where $\alpha_u \in R^k$ ($u \in \overline{1, m}$) and α_u represents the latent factor vector of user u. Similarly, we denote $P^T = (\beta_1, \beta_2, ..., \beta_n)$ where $\beta_p \in R^k$ ($p \in \overline{1, n}$) and β_p represents the latent factor vector of item p. The objective of WMF is defined by:

$$\mathcal{L}_{WMF} = \frac{1}{2}\sum_{u,p} w_{up}(M_{up} - \alpha_u^T\beta_p)^2 + \frac{1}{2}\left(\lambda_\alpha \sum_u ||\alpha_u||^2 + \lambda_\beta \sum_p ||\beta_p||^2\right) \tag{4.1}$$

where w_{up} is a hyperparameter to compensate the interaction between user u and item p, and is used to balance between the number of non-zero and zero values in a sparse user-item matrix. The weight w of the interaction between user u and item p (denoted as w_{up}) can be set as $w_{up} = l(1+\phi M_{up})$ [23, 76] where l is a relative scale and ϕ is a constant. λ_α and λ_β are used to adjust the importance of two quadratic regularization terms $\sum_u ||\alpha_u||^2$ and $\sum_p ||\beta_p||^2$.

Word embedding models. Word embedding models have recently received a lot of attention from the research community. Given a sequence of training words, the embedding models learn a latent representation for each word. For example, *word2vec* [77] is one of popular word embedding methods. Especially, the skip-gram model in word2vec tries to predict surrounding words (i.e., word context) of a given word in the training set.

According to Levy et al. [78], skip-gram model with negative sampling (SGNS) is equivalent to implicitly factorize a word-context matrix, whose cells are the *Pointwise Mutual Information* (PMI) of the respective word and context pairs, shifted by a global constant. Let D as a collection of observed word and context pairs, the PMI between a word i and its word context j is calculated as:

$$PMI(i, j) = log\frac{P(i, j)}{P(i) * P(j)}$$

where $P(i, j)$ is the joint probability that word i and word j appears together within a window size (e.g. $P(i,j) = \frac{\#(i,j)}{|D|}$, where $|D|$ refers to the total number of word and word context pairs in D). Similarly, $P(i)$ is the probability the word i appears in D, and $P(j)$ is the probability word j appears in D (e.g. $P(i) = \frac{\#(i)}{|D|}$ and $P(j) = \frac{\#(j)}{|D|}$). Obviously, $PMI(i,j)$ can be calculated as:

$$PMI(i,j) = log\frac{\#(i,j) * |D|}{\#(i) * \#(j)} \tag{4.2}$$

By calculating PMI of all word-context pairs in D, we can form a squared $n \times n$ matrix M^{PMI} where n is the total number of distinct words in D. Next, a *Shifted Positive Pointwise Mutual Information* (SPPMI) of two words i and j is calculated as:

$$SPPMI(i,j) = max(PMI(i,j) - log(s), 0) \tag{4.3}$$

where s is a hyperparameter to control the density of PMI matrix M^{PMI} and s can be interpreted equivalently as a hyperparameter that indicates the number of negative samples in SGNS. When s is large, more values in the matrix M^{PMI} are cleared, leading M^{PMI} to become sparser. When s is small, matrix M^{PMI} becomes denser. Finally, factorizing matrix M^{SPPMI}, where each cell in M^{SPPMI} is transformed by Formula (4.3), is equivalent to performing SGSN.

Co-liked item embedding (LIE). As mentioned in the previous studies [76, 79, 80], when users liked/consumed items in a sequence, the items sorted by the ascending interaction time order can be inferred as a sequence. Thus, performing co-liked item embeddings to learn latent representations of items is equivalent to perform word embeddings to learn latent representations of words. Therefore, we can apply word embedding methods to learn latent representations of items, and perform a joint learning between embedding models and traditional factorization methods (e.g. WMF).

Given each user's liked item list, we generate co-liked item-item co-occurrence pairs without considering liked time. Particularly, given a certain item in the item sequence, we consider all other items as its contexts. We call this method as a *greedy context generation* method which can be applied to other non-timestamped datasets. After generating item and item context pairs, we construct an item co-occurrence SPPMI matrix and perform SPPMI matrix factorization. In particular, given generated item-item co-occurrence pairs, we construct a SPPMI matrix of items by applying Equation (4.2) to calculate the point-wise mutual information of each pair, and then by measuring the shifted positive point-wise mutual information of the pair based on Equation (4.3). Once the SPPMI matrix of co-liked items is constructed, we incorporate it to the traditional matrix factorization method to improve the item latent representations.

Co-disliked item embedding (DIE). As mentioned in the Introduction section, when many users disliked two items p_1 and p_2 together, the two items can form a pair of co-occurred disliked items. If the recommender systems learned this disliked co-occurrence pattern, it would not recommend item p_2 to a user, who disliked p_1. This will help reduce the false positive rate for the recommender systems. Therefore, similar to liked item embeddings, we applied the word embedding technique to exploit the disliked co-occurrence information to enhance the item's latent factors.

User embedding (UE). When two users A and B preferred same items, we can assume the two users share similar interests. Therefore, if user A enjoyed an item p that has not been observed in user B's transactions, we can recommend the item to user B. Similar to liked and disliked item embeddings, we applied the word embedding technique to learn user embeddings that explain the co-occurrence patterns among users.

From the user-item interaction matrix $M^{m \times n}$, where each row represents consumed items of a user (e.g. a list of items that the user rated or backed), we only keep liked items per user in the matrix M'. Then, we construct a $n \times m$ reverse matrix M'^T of M', where each row represents users that liked a certain item. Then, users, who liked the same item, form a sequence, and the sequence of users is interpreted as a sequence of words. From this point, word embedding techniques are applied to the user sequence to enhance latent representations of users.

Our RME model. It is a joint learning model combining WMF, co-liked item embedding, co-disliked item embedding, and user embedding. It minimizes the following objective function:

$$
\begin{aligned}
\mathcal{L} = &\overbrace{\frac{1}{2}\sum_{u,p} w_{up}(M_{up} - \alpha_u^T\beta_p)^2}^{\mathcal{L}_1} \text{ (WMF)} \\
+ &\overbrace{\frac{1}{2}\sum_{X_{pi}\neq 0} w_{pi}^{(+p)}(X_{pi} - \beta_p^T\gamma_i - b_p - c_i)^2}^{\mathcal{L}_2} \text{ (LIE)} \\
+ &\overbrace{\frac{1}{2}\sum_{Y_{pi\prime}\neq 0} w_{pi\prime}^{(-p)}(Y_{pi\prime} - \beta_p^T\delta_{i\prime} - d_p - e_{i\prime})^2}^{\mathcal{L}_3} \text{ (DIE)} \\
+ &\overbrace{\frac{1}{2}\sum_{Z_{uj}\neq 0} w_{uj}^{(u)}(Z_{uj} - \alpha_u^T\theta_j - f_u - g_j)^2}^{\mathcal{L}_4} \text{ (UE)} \\
+ &\frac{1}{2}\lambda\left(\sum_u ||\alpha_u||^2 + \sum_p ||\beta_p||^2 + \sum_i ||\gamma_i||^2 + \sum_{i\prime} ||\delta_{i\prime}||^2 + \sum_j ||\theta_j||^2\right)
\end{aligned}
\tag{4.4}
$$

where the item's latent representation β_p is shared among WMF, co-liked item embedding and co-disliked item embedding. The user's latent representation α_u is shared between WMF and user embedding. X and Y are SPPMI matrices, constructed by co-liked item-item co-occurrence patterns and disliked item-item co-occurrence patterns, respectively. γ and δ are $k \times 1$ latent representation vectors of co-liked item context and co-disliked item context, respectively. Z is a SPPMI matrix constructed by user-user co-occurrence patterns. θ is a $k \times 1$ latent representation vector of a user context. $w^{(+p)}$, $w^{(-p)}$ and $w^{(u)}$ are hyperparameters to compensate for item/user co-occurrences in X, Y and Z when performing decomposition. b is liked item bias, and c is co-liked item context bias. d is disliked item bias, and e is co-disliked item-context bias. f and g are user bias and user context bias, respectively. Incorporating bias terms were originally introduced in [81]. A liked item bias b_p and a co-liked item context bias c_i mean that when the two items p_i and p_j are co-liked by users, each item may have a little bit higher/lower preference compared to the average preference. The similar explanation is applied to the other biases. The last line shows regularization terms along with a hyperparameter λ to control their effects.

Optimization We can use the stochastic gradient descent to optimize the Equation (4.4). However, it is not stable and sensitive to parameters [82]. Therefore, we adopt vector-wise ALS algorithm [82, 83] that alternatively optimize each model's parameter in parallel while fixing the other parameters until the model gets converged. Specifically, we calculate the partial derivatives of the model's objective function with regard to the model parameters (i.e., $\{\alpha_{1:m}, \beta_{1:n}, \gamma_{1:n}, \delta_{1:n}, b_{1:n}, c_{1:n}, d_{1:n}, e_{1:n}, \theta_{1:m}, f_{1:m}, g_{1:m}\}$). Then we set them to zero and obtain updating rules. Details are given as follows:

From the objective function in Equation (4.4), while taking partial derivatives of $\mathcal{L}$ with regard to each user's latent representation vector α_u, we observe that only $\mathcal{L}_1$, $\mathcal{L}_4$ and the L2 user regularization $\frac{1}{2}\lambda\sum_u ||\alpha_u||^2$ contain α_u. Therefore, we obtain:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\alpha_u} &= \frac{\partial\mathcal{L}_1}{\partial\alpha_u} + \frac{\partial\mathcal{L}_4}{\partial\alpha_u} + \frac{\partial\lambda\sum_u ||\alpha_u||^2}{2\partial\alpha_u} \\ &= -\sum_{u,p} w_{up}(M_{up} - \alpha_u^T\beta_p)\beta_p^T - \sum_{u,j} w_{uj}^{(u)}(Z_{uj} - \alpha_u^T\theta_j - f_u - g_j)\theta_j^T + \lambda\alpha_u^T\end{aligned}$$

Fixing item latent vectors β, user context latent vectors θ, user bias d and user context bias e, and solving $\frac{\partial\mathcal{L}}{\partial\alpha_u} = 0$, we obtain the updating rule of α_u as follows:

$$\begin{aligned}\alpha_u =& \Big[\sum_p w_{up}\beta_p\beta_p^T + \sum_{j|Z_{uj}\neq 0} w_{uj}^{(u)}\theta_j\theta_j^T + \lambda I_K\Big]^{-1}\\ &\Big[\sum_p w_{up}M_{up}\beta_p + \sum_{j|Z_{uj}\neq 0} w_{uj}^{(u)}(Z_{uj} - f_u - g_j)\theta_j\Big]\\ &,\forall 1 \leq u \leq m, 1 \leq p \leq n, 1 \leq j \leq m\end{aligned} \tag{4.5}$$

Similarly, taking partial derivatives of $\mathcal{L}$ with respect to each item latent vector β_p needs to consider only $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ and item regularization $\frac{1}{2}\lambda\sum_p ||\beta_p||^2$. By fixing other parameters and solving $\frac{\partial\mathcal{L}}{\partial\beta_p} = 0$, we obtain:

$$\begin{aligned}\beta_p =& \Big[\sum_u w_{up}\alpha_u\alpha_u^T + \sum_{i|X_{pi}\neq 0} w_{pi}^{(+p)}\gamma_i\gamma_i^T + \sum_{i\prime|Y_{pi\prime}\neq 0} w_{pi\prime}^{(-p)}\delta_{i\prime}\delta_{i\prime}^T + \lambda I_K\Big]^{-1}\\ &\Big[\sum_u w_{up}M_{up}\alpha_u + \sum_{i|X_{pi}\neq 0} w_{pi}^{(+p)}(X_{pi} - b_p - c_i)\gamma_i +\\ &\sum_{i\prime|Y_{pi\prime}\neq 0} w_{pi\prime}^{(-p)}(Y_{pi\prime} - d_p - e_{i\prime})\delta_{i\prime}\Big]\\ &,\forall 1 \leq u \leq m, 1 \leq p \leq n, 1 \leq i, i' \leq n\end{aligned} \tag{4.6}$$

In the same manner, we obtain the update rules of item contexts γ, δ, and user context θ alternatively as follows:

$$\begin{aligned}\gamma_i =& \Big[\sum_{p|X_{ip}\neq 0} w_{ip}^{(+p)}\beta_p\beta_p^T + \lambda I_K\Big]^{-1}\Big[\sum_{p|X_{ip}\neq 0} w_{ip}^{(+p)}(X_{ip} - b_p - c_i)\beta_p\Big]\\ \delta_{i\prime} =& \Big[\sum_{p|Y_{i\prime p}\neq 0} w_{i\prime p}^{(-p)}\beta_p\beta_p^T + \lambda I_K\Big]^{-1}\Big[\sum_{p|Y_{i\prime p}\neq 0} w_{i\prime p}^{(-p)}(Y_{i\prime p} - d_p - e_{i\prime})\beta_p\Big]\\ \theta_j =& \Big[\sum_{u|Z_{ju}\neq 0} w_{ju}^{(u)}\alpha_u\alpha_u^T + \lambda I_K\Big]^{-1}\Big[\sum_{u|Z_{ju}\neq 0} w_{ju}^{(u)}(Z_{ju} - d_u - e_j)\alpha_u\Big]\\ &,\forall 1 \leq u \leq m, 1 \leq p \leq n, 1 \leq i, i' \leq n\end{aligned} \tag{4.7}$$

The item biases and item context biases b, c, d, e, as well as the user and user context biases f, g are updated alternatively using the following update rules:

$$\begin{aligned}
b_p =& \frac{1}{|i : X_{pi} \neq 0|} \sum_{i:X_{pi}\neq 0} (X_{pi} - \beta_p^T \gamma_i - c_i) \\
c_i =& \frac{1}{|p : X_{ip} \neq 0|} \sum_{p:X_{ip}\neq 0} (X_{ip} - \beta_p^T \gamma_i - b_p) \\
d_p =& \frac{1}{|i\prime : Y_{pi\prime} \neq 0|} \sum_{i\prime:Y_{pi\prime}\neq 0} (Y_{pi\prime} - \beta_p^T \delta_{i\prime} - e_{i\prime}) \\
e_{i\prime} =& \frac{1}{|p : Y_{i\prime p} \neq 0|} \sum_{p:Y_{i\prime p}\neq 0} (Y_{i\prime p} - \beta_p^T \delta_{i\prime} - d_p) \\
f_u =& \frac{1}{|j : Z_{uj} \neq 0|} \sum_{j:Z_{uj}\neq 0} (Z_{uj} - \alpha_u^T \theta_j - g_j) \\
g_j =& \frac{1}{|u : Z_{ju} \neq 0|} \sum_{u:Z_{ju}\neq 0} (Z_{ju} - \alpha_u^T \theta_j - f_u)
\end{aligned} \tag{4.8}$$

In short, the pseudocode of our proposed RME model is presented in Algorithm 1.

```
Require: M, $\lambda$
1: Build SPPMI matrices of liked item $X$, disliked item $Y$ and user co-occurrences $Z$ using Eq. (4.2) and Eq. (4.3)
2: Initialize $U$ (or $\alpha_{1:m}$), $P$ (or $\beta_{1:n}$), $\gamma_{1:n}, \delta_{1:n}, \theta_{1:m}$.
3: Initialize $b_{1:n}, c_{1:n}, d_{1:n}, e_{1:n}, f_{1:m}, g_{1:m}$.
4: repeat
5:     For each user u, update $\alpha_u$ by Eq. (4.5) ($1 \leq u \leq m$).
6:     For each item p, update $\beta_p$ by Eq. (4.6) ($1 \leq p \leq n$).
7:     Alternatively update each item context $\gamma_i$, $\delta_{i'}$ and user context $\theta_j$ by Eq. (4.7) ($1 \leq i, i' \leq n; 1 \leq j \leq m$).
8:     Alternatively update each bias $b_p, c_i, d_p, e_{i'}, f_u, g_j$ by Eq. (4.8) ($1 \leq p, i, i' \leq n$; $1 \leq u, j \leq m$).
9: until convergence
10: return $U, P$
```

Algorithm 1: RME algorithm

Complexity Analysis In this section, we briefly provide time complexity analysis of our model. Let Ω_M = {(u, p) — $M_{up} \neq 0$} , Ω_X = {(p, i) — $X_{pi} \neq 0$ }, Ω_Y = {(p, i') — $Y_{pi'} \neq 0$}, Ω_Z = {(u, j) — $Z_{uj} \neq 0$}. Constructing SPPMI matrices X, Y and Z take $O(|\Omega_X|^2)$, $O(|\Omega_Y|^2)$ and $O(|\Omega_Z|^2)$, respectively. However, the SPPMI matrices are calculated once and are constructed in parallel using batch processing, so they are not costly. For learning RME model, computing α takes $O((|\Omega_M| + |\Omega_Z|)k^2 + k^3)$ time, and computing β takes $O((|\Omega_M| + |\Omega_X| + |\Omega_Y|)k^2 + k^3)$ time. Also, it takes $O(|\Omega_X|k^2 + k^3)$ for computing co-liked item context γ, and so do other latent contexts δ, θ. It takes $O(|\Omega_Z|k)$ time to compute all user bias f and so do the other biases. Thus, the time complexity for RME is $O(\eta(2(|\Omega_M| + |\Omega_X| + |\Omega_Y| + |\Omega_Z|)k^2 + (2m + 3n)k^3))$, where η is the number of iterations. Since $k << min(m, n)$ and M, X, Y, Z are often sparse, which mean $(|\Omega_M| + |\Omega_X| + |\Omega_Y| + |\Omega_Z|)$ is small, the time complexity of RME is shortened as $O(\eta(m + \frac{3}{2}n)k^3)$, which scales linearly to the conventional ALS algorithm for collaborative filtering [82].

Inferring Disliked Items in Implicit Feedback datasets Unlike explicit feedback datasets, there is a lack of substantial evidence, on which items the users disliked in implicit feedback datasets. Since our model exploits co-disliked item co-occurrences patterns among items, the implicit feedback datasets challenge our model. To deal with it, we can simply assume that missing values are equally likely to be negative feedback, then sample some negative instances from missing values with uniform weights [26, 72, 73, 84]. However, assigning uniform weight is suboptimal because the missing values are a mixture of negative and unknown feedbacks. A recent work suggests to sample negative instances by assigning non-uniform weights based on item popularity [74]. The idea is that popular items are highly aware by users, so if they are not observed in a user's transactions, it assumes that the user dislikes them. However, this sampling method is also not optimal because same unobserved popular items can be sampled across multiple users. This approach does not reflect user's personalized interests.

Instead, we follow the previous works [85, 86, 87], and propose a user-oriented EM-like algorithm to draw negative samples (i.e., inferred disliked items) for users in implicit feedback datasets. Our approach is described as follows:

First, we assume that an item with a low ranking score of being liked will have a higher probability to be drawn as a negative sample of a user. Given r_u is the ranked list of all items of the user u, the prior probabilities of items to be drawn as negative samples are calculated by using a softmax function as follows:

$$Pr_i^{(u)} = \frac{\exp\left(-r_u[i]\right)}{\sum_{j=1}^{n} \exp\left(-r_u[j]\right)} \tag{4.9}$$

After negative samples are drawn for each user, we built the RME model by using Algorithm 1. The pseudocode of the RME model for implicit feedback datasets is presented in Algorithm 2.

In Algorithm 2, since each user may prefer a different number of items, we define a hyper-parameter τ as a negative sample drawing ratio to control how many negative samples we will sample for each user. In line 6, $count(u)$ returns the number of observed items of a user u. Then, the number of drawn negative samples for the user u is calculated and assigned to ns. If a user prefers 10 items and $\tau = 0.8$, the algorithm will sample 8 disliked items. We note that sampling with replacement is used such that different items are drawn independently. The value of τ is selected using the validation data. In line 8, we set the ranking of observed items to $+\infty$ to avoid drawing the observed items as negative samples. In line 12, we build the RME model based on the negative samples drawn in the Expectation step, and temporally store newly learned user latent matrix, item latent matrix and corresponding NDCG to $U\_tmp, P\_tmp, ndcg$ variables, respectively (NDCG is a measure to evaluate recommender systems, which will be mentioned in Experiment section). If we obtain a better $ndcg$ comparing with the previous NDCG $prev\_ndcg$ (line 13), we will update $U, P, prev\_ndcg$ with new values (line 14). Overall, at the end of the Expectation step, we obtain the disliked items for each user. Then, in the Maximization

```
Require:   M, negative sample drawing ratio $\tau$
 1: $max\_iter$ = 10, $prev\_ndcg$ = 0, $iter$ = 0
 2: Initialize Step: $U, P = WMF(M)$
 3: repeat
 4:     $iter$ += 1
 5:                                                        ▷ Expectation Step
 6:     for $u \in [1, m]$ do:
 7:         $ns$ = $\tau$ * $count$(u)
 8:         Compute ranked item list: $r_u = P.\alpha_u$
 9:         Assign observed items with ranking of $+\infty$.
10:         Measure prior probabilities of items to be drawn as negative samples by Eq.
    (4.9) then randomly draw $ns$ negative samples with those prior probabilities.
11:     end for
12:                                     ▷ Maximization Step with early stopping
13:     $U\_tmp, P\_tmp, ndcg$ = RME(train_data, vad_data)
14:     if $ndcg > prev\_ndcg$ then
15:         $U, P, prev\_ndcg = U\_tmp, P\_tmp, ndcg$
16:     else
17:         break                                          ▷ Early stopping
18:     end if
19: until $iter$ ¡ $max\_iter$
20: return $U, P$
```

Algorithm 2: RME model for implicit feedback datasets using user-oriented EM-like algorithm to draw negative samples

step, we build our RME model to re-learn user and item latent representations U and P. The process is repeated until getting converged or the early stopping condition (line 13 to 17) is satisfied.

Time Complexity: In order to construct RME model for implicit feedback datasets, we need to re-learn RME model, which includes re-building 3 SPPMI matrices in the maximization step in $\eta\prime$ iterations to get converged. Thus, it takes $O(\eta\prime((|\Omega_X|^2 + |\Omega_Y|^2 + |\Omega_Z|^2) + \eta(m + \frac{3}{2}n)k^3))$ time where $\eta\prime$ is small.

Table 4.2: Performance of the baselines, our RME model, and its two variants. The improvement of our model over the baselines and its variants were significant with *p-value* $<$ 0.05 in the three datasets under the non-directional two-sample t-test.

Method	**MovieLens-10M**			**MovieLens-20M**			**TasteProfile**		
	Recall@5	NDCG@20	MAP@10	Recall@5	NDCG@20	MAP@10	Recall@5	NDCG@20	MAP@10
Item-KNN	0.0137	0.0338	0.0397	0.0131	0.0345	0.0402	0.0793	0.0685	0.0904
Item2vec	0.1020	0.1001	0.0502	0.1066	0.1019	0.0539	0.1455	0.1593	0.0727
WMF	0.1280	0.1245	0.0655	0.1348	0.1290	0.0720	0.1745	0.1853	0.0931
Cofactor	0.1460	0.1381	0.0772	0.1480	0.1387	0.0804	0.1771	0.1873	0.0950
U_RME	0.1516	0.1412	0.0818	0.1524	0.1425	0.0847	0.1825	0.1899	0.0997
I_RME	0.1511	0.1422	0.0817	0.1530	0.1412	0.0838	0.1826	0.1915	0.0996
RME	**0.1562**	**0.1458**	**0.0841**	**0.1570**	**0.1461**	**0.0869**	**0.1876**	**0.1954**	**0.1025**

4.2.2 Experimental Settings

Datasets: To measure the performance of our RME model, we evaluate the model on 3 real-world datasets:

- MovieLens-10M [68]: is an explicit feedback dataset. It consists of 69,878 users and 10,677 movies with 10m ratings. Following the k-cores preprocessing [26, 88], we only kept users, who rated at least 5 movies, and movies, which were rated by at least 5 users. This led to 58,057 users and 7,223 items (density= 0.978%).
- MovieLens-20M: is an explicit feedback dataset. It consists of 138,000 users, 27,000 movies, and 20 millions of ratings. We filtered with the same condition as for MovieLens-10M. This led to 111,146 users and 9,888 items (density= 0.745%).
- TasteProfile: is an implicit feedback dataset containing a song's play count by a user [1]. The play counts are user's implicit preference and are binarized. Similar to [76], we first subsampled the dataset to 250k users and 25k items. Then we kept only users, who listened to at least 20 songs, and songs, which were listened by at least 50 users. As a result, 221,011 users and 22,713 songs were remained (density= 0.291%).

[1]http://the.echonest.com/

Baselines: To illustrate the effectiveness of our RME model, we compare it with the following baselines:

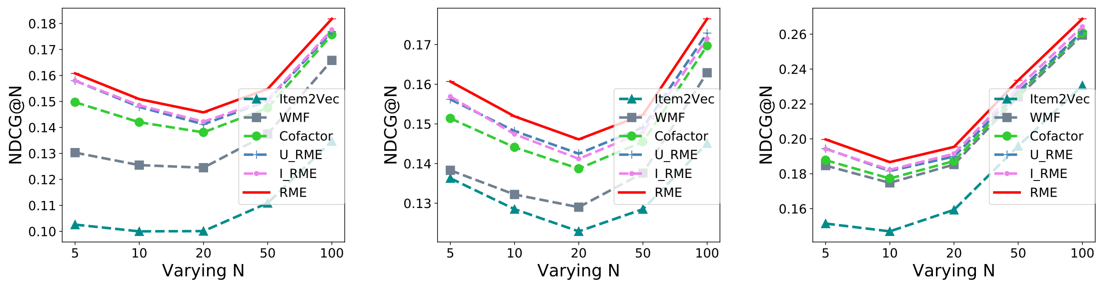
- WMF [23]: It is a weighted matrix factorization with *l*2-norm regularization.
- Item-KNN [89]: This is an item neighborhood-based collaborative filtering method.
- Item2Vec [80]: This method used Skip-gram with negative sampling [77] to learn item embeddings, then adopted a similarity score between item embeddings to generate user's recommendation lists.
- Cofactor [76]: This is a method that combines WMF and co-liked item embedding.

We note that we do not compare our models with user collaborative filtering method (i.e. User-KNN) because it is not applicable to run the method on the large datasets. However, [90] reported that User-KNN had worse performance than Item-KNN, especially when there are many items but few ratings in a dataset.

Our models: We not only compare the baselines with our RME, but also two variants of our model such as U_RME and I_RME to show the effectiveness of incorporating all of the user embeddings, liked-item embeddings and disliked-item embeddings:

- U_RME (i.e., RME - DIE): This is a variant of our model, considering only WMF, user embeddings, and liked-item embeddings.
- I_RME (i.e., RME - UE): This is another variant of our model, considering only WMF, liked-item embeddings, and disliked-item embeddings.
- RME: This is our proposed RME model.

Evaluation metrics. We used three well-known ranking-based metrics – Recall@N, normalized discounted cumulative gain (NDCG@N), and mean average precision (MAP@N).



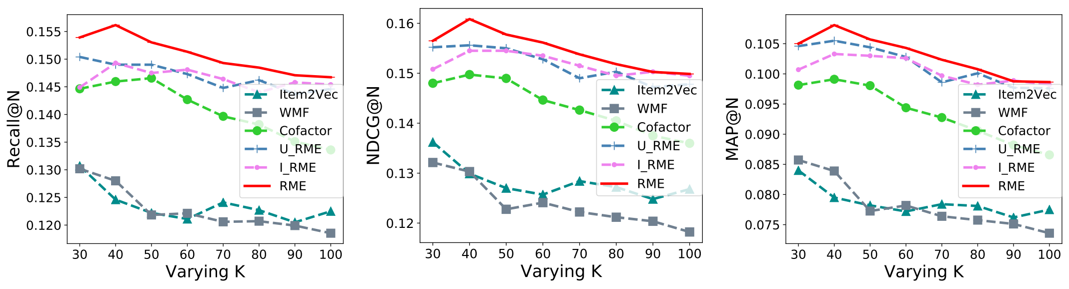
(a) NDCG@N on MovieLens-10M. (b) NDCG@N on MovieLens-20M. (c) NDCG@N on TasteProfile.

Figure 4.2: Performance of all models when varying top N.

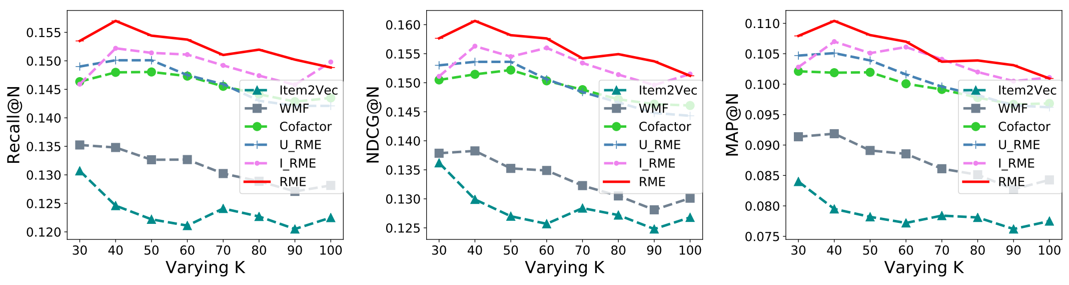
Recall@N considers all items in top N items equally, whereas NDCG@N and MAP@N apply an increasing discount of log_2 to items at lower ranks.

Training, validation and test sets. We follow 70/10/20 proportions for splitting the original dataset into training/validation/test sets [91]. MovieLens-10M and MovieLens-20M datasets contain timestamp values of user-movie interactions. To create training, validation and testing sets for these datasets, we sorted all user-item interaction pairs in the ascending interaction time order in each of MovieLens-10M and MovieLens-20M datasets. The first 80% was used for training and validation, and the rest 20% data was used as a test set. Out of 80% data extracted for training and validation, we randomly took 10% for the validation set. To measure the statistical significance of RME over the baselines, we repeated the splitting process five times (i.e., generating five pairs of training and validation sets). Since TasteProfile dataset did not contain timestamp information of user-song interactions, we randomly split the TasteProfile dataset into training/validation/test sets five times with 70/10/20 proportions. Averaged results are reported in the following subsection.

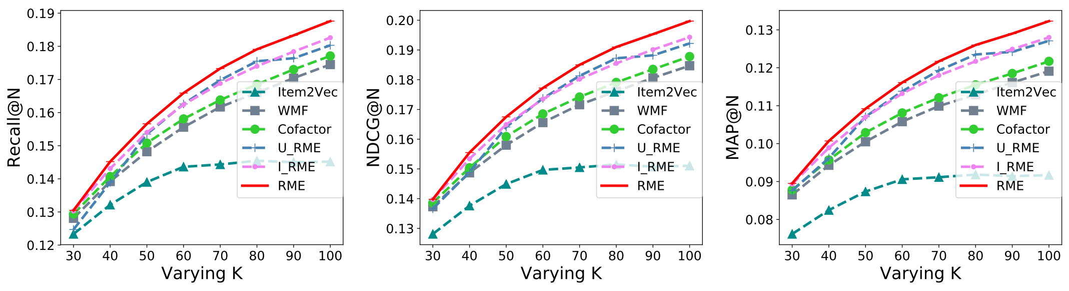
Stopping criteria and Hyperparameters. To decide when to stop training a model, we measured the model's $NDCG@100$ by using the validation set. We stopped training the model when there was no further improvement. Then, we applied the best model to the



(a) Recall@5, NDCG@5, MAP@5 on MovieLens-10M. Fix $\lambda = 1$, and vary k.



(b) Recall@5, NDCG@5, MAP@5 on MovieLens-20M. Fix $\lambda = 0.5$, and vary k.



(c) Recall@5, NDCG@5, MAP@5 on TasteProfile. Fix $\lambda = 10$, $\tau = 0.2$, and vary k.

Figure 4.3: Performance of models when varying the latent dimension size k with fixing the value of λ.

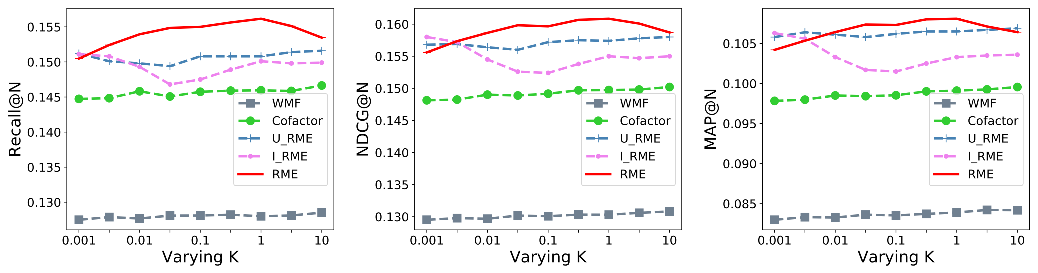
test set to evaluate its performance. This method was applied to the baselines and RME.

All hyper-parameters were tuned on the validation set by a grid search. We used the same hyper-parameter setting in all models. The grid search of the regularization weight λ was performed in $\{0.001, 0.005, 0.01, 0.05, ..., 10\}$. The size of latent dimensions was in a range of $\{30, 40, 50, ..., 100\}$. We set weights $w^{(+p)} = w^{(-p)} = w^{(u)} = w$ for all user-user and item-item co-occurrence pairs. When building our RME model for TasteProfile dataset, we do a grid search for the negative sample drawing ratio τ in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$.

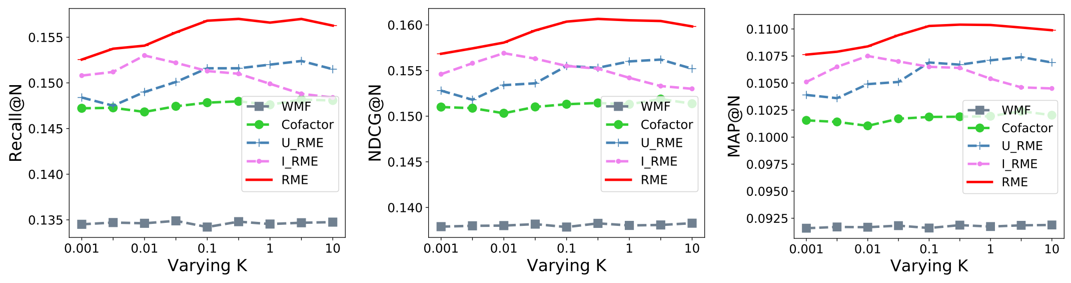
4.2.3 Experimental Results

Performance of the baselines and RME. Table 4.2 presents recommendation results of RME and compared models at Recall@5, NDCG@20, and MAP@10. First, we compared RME with the baselines. We observed that RME outperformed all baselines in the three datasets, improving the Recall by 6.3%, NDCG by 5.1%, and MAP by 8.3% on average over the best baseline (*p-value* $<$ 0.001). Second, we compared two variants of RME model with the baselines. We see that both U_RME and I_RME performed better than the baselines. Adding user embeddings improved the Recall by 3.0$\sim$3.5%, NDCG by 1.4$\sim$2.2%, and MAP by 4.2$\sim$5.8% (*p-value* $<$ 0.001), while adding disliked item embeddings improved the Recall by 3.1$\sim$3.8%, NDCG by 2.2$\sim$3.0%, and MAP by 4.9$\sim$6.0%. Third, we compare RME with its two variants. RME also achieved the best result, improving Recall by 2.6$\sim$3.0%, NDCG by 2.0$\sim$2.5%, and MAP by 2.6$\sim$2.8% (*p-value* $<$ 0.05). We further evaluated NDCG@N of our model when varying top N in range $\{5, 10, 20, 50, 100\}$. Figure 4.2 shows our result (we excluded Item-KNN in the figure and following figures since it performed extremely worst). Our model still performed the best. On average, it improved NDCG@N by 6.2% comparing to the baselines, and by 3.3% comparing to its variants. These experimental results show that both co-disliked item embedding and user embedding positively contributed to RME.

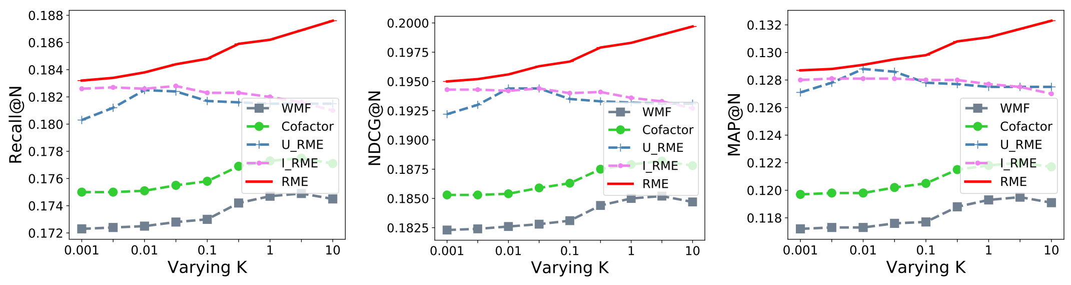
The experimental results in TasteProfile in Table 4.2 showed that inferring disliked items in Algorithm 2 worked well since RME model incorporating co-disliked item embedding outperformed the baselines. To further confirm the effectiveness of the algorithm, we also applied it to MovieLens-10M and MovieLens-20M datasets after removing the explicit disliking information, pretending them as implicit feedback datasets. In the datasets without disliking information, RME under Algorithm 2 still outperformed the best baseline with 4.2%, 4.6% and 7.2% improvements on average in Recall, NDCG and MAP, respectively (*p-value* ¡ 0.001). Its performance was slightly lower than the original



(a) Recall@5, NDCG@5, MAP@5 on MovieLens-10M. Fix $k = 40$, and vary λ.



(b) Recall@5, NDCG@5, MAP@5 on MovieLens-20M. Fix $k = 40$, and vary λ.



(c) Recall@5, NDCG@5, MAP@5 on TasteProfile. Fix $k = 100$, $\tau = 0.2$, and vary λ.

Figure 4.4: Performance of models when varying λ with fixing the latent dimension size k. Item2Vec did not contain regularization, so we excluded it.

RME (based on explicit disliking information) at 0.3%, 0.7% and 1.3% on average in Recall, NDCG, and MAP, respectively. The experimental results confirmed the effectiveness of Algorithm 2. We note that Algorithm 2 got converged in up to 4 iterations for all three datasets by the early stopping condition. Due to the space limitation, we do not include figures which show the loss over iterations.

Parameter sensitivity analysis: We analyze the effects of the parameters in RME model in order to answer the following research questions: (*RQ2-1*:) How does RME work when varying the latent dimension size k?; (*RQ2-2*:) How does RME model change with varying λ?; (*RQ2-3*:) How sensitive is the RME model on an implicit feedback dataset (e.g. TasteProfile) when varying negative sample drawing ratio τ?; and (*RQ2-4*:) Can RME achieve better performance with a dynamic setting of regularization hyper-parameters?

Regarding *RQ2-1*, Figure 4.3 shows the sensitivity of all compared models when fixing λ and varying the latent dimension size k in {30, 40, 50, 60, 70, 80, 90, 100}. It is clearly observed that our model outperforms the baselines in all datasets. In MovieLens-10M and MovieLens-20M datasets, all six models downgrade the performance when the latent dimension size k is over 60. In the TasteProfile dataset, when increasing k, although all models gain a higher performance, our model tends to achieve much higher performance.

In a *RQ2-2* experiment, we exclude Item2Vec because this model does not contain the regularization term. We fix $k = 40$ in MovieLens-10M and MovieLens-20M. In TasteProfile dataset, we fix k=100, τ=0.2. We vary lambda in range {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10}. Then, we report the average results of Recall@5, NDCG@5, and MAP@5. As shown in Figure 4.4, the performance of our model is better than the baselines. In MovieLens-10M and MovieLens-20M dataset, RME increases its performance when increasing λ up to 1, then its performance goes down when λ is increasing

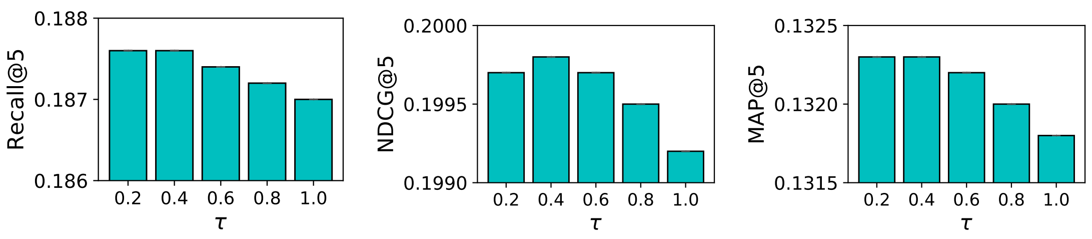
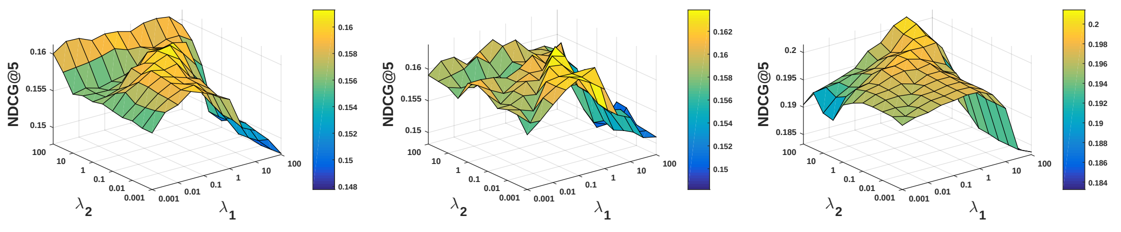


Figure 4.5: Performance of RME in TasteProfile when varying negative sample drawing ratio τ with fixing $k = 100$, $\lambda = 10$.

more. In TasteProfile, RME tends to gain a higher performance and more outperformed the baselines when λ is increasing.

To understand the sensitivity of our model when varying negative sample drawing ratio τ in the implicit feedback dataset – TasteProfile (RQ2-3), we vary τ in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, and fix $k = 100$ and $\lambda = 10$. Figure 4.5 shows that when τ increases, our model degrades with a small amount (e.g. around -0.3% in Recall@5 and NDCG@5, and -0.4% in MAP@5). In NDCG@5, our model gains the best result when $\tau = 0.4$. We note that our worst case (when $\tau = 1.0$) is still better than the best baseline presented in Table 4.2. This shows that the sensitivity of our model with regard to the negative sample drawing ratio τ is small/limited.

In our previous experiments, we used a static setting of regularization hyper-parameters by setting $\lambda_\alpha = \lambda_\beta = \lambda_\gamma = \lambda_\delta = \lambda_\theta = \lambda$. To explore if a dynamic setting of those regularization hyper-parameters could lead to better results for RME model (RQ2-4), we set $\lambda_\alpha = \lambda_\beta = \lambda_1$, $\lambda_\gamma = \lambda_\delta = \lambda_\theta = \lambda_2$. Then we both vary λ_1 and λ_2 in $\{100, 50, 10, 5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$ while fixing the latent dimension size k. Next, we report the NDCG@5 for all 3 datasets. As shown in Figure 4.6, our model even get a higher performance with the dynamic setting. For example, it gains NDCG@5 = 0.1613 when $\lambda_1 = 100$ and $\lambda_2 = 0.005$ in MovieLens-10M dataset. Similarly, NDCG@5 = 0.1639 when $\lambda_1 = 0.5$, $\lambda_2 = 1$ in MovieLens-20M dataset. NDCG@5 = 0.2014 when $\lambda_1 = 100$, $\lambda_2 = 10$ in TasteProfile dataset. The dynamic setting produced 0.3$\sim$2% higher



(a) MovieLens-10M (b) MovieLens-20M (c) TasteProfile

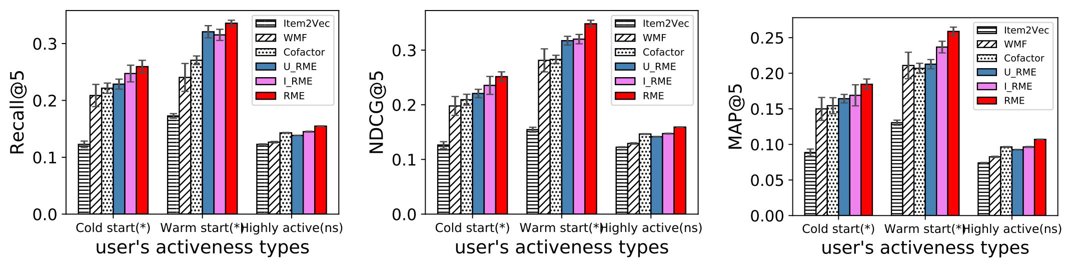
Figure 4.6: Performance of RME under a dynamic setting of regularization hyper-parameters. Set $\lambda_\alpha = \lambda_\beta = \lambda_1$, and $\lambda_{\gamma(+)} = \lambda_{\gamma(-)} = \lambda_\theta = \lambda_2$.

results than the static setting presented in Table 4.2.

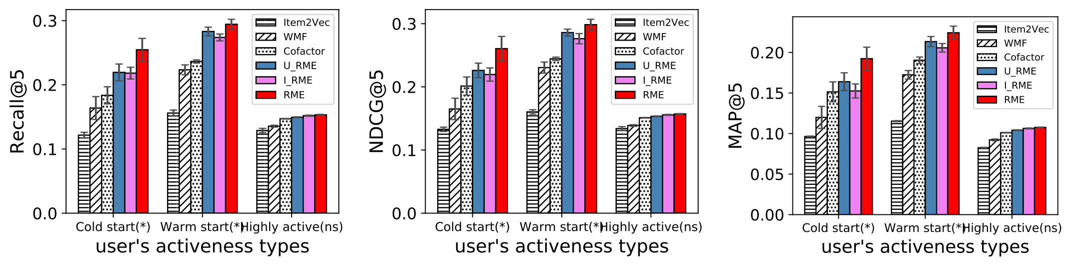
So far, we compared the performance of our model and the baselines while varying values of hyper-parameters. We showed that our model outperformed the baselines in all cases, indicating that our model was less sensitive with regard to the hyper-parameters. We also showed that our model produced better results under the dynamic setting.

Performance of models for different types of users. We sorted users by the ascending order of their activity level in terms of the number of liked items. Then we categorized them into three groups: (1) *cold-start users* who were in the first 20% of the sorted user list (i.e., their activity level is the lowest); (2) *warm-start users* who were in between 20% and 80% of the sorted user list; (3) *highly active users* who were in the last 20% of the sorted user list (i.e., the most active users). Then, we measured the performance of all the compared models for each of the user groups.

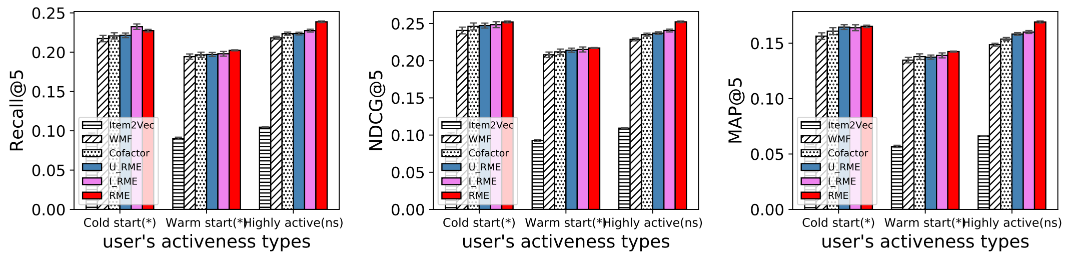
Figure 4.7 shows the performance of all the compared models in MovieLens-10M, MovieLens-20M and TasteProfile datasets. In MovieLens-10M (Figure 4.7(a)), our model significantly outperformed the baselines and the two variants in all three user groups, improving Recall@5 by 4.7∼6.7%, NDCG@5 by 6.8∼8.8%, and MAP@5 by 9.2∼11.0% over the best compared method. In MovieLens-20M dataset (Figure 4.7(b)), our model significantly outperformed the baselines and its variants in 2 groups: *cold-start users* and *warm-start users*. It improved Recall@5 by 16.1%, 4.0%, 0.8%, NDCG@5 by 15.3%,



(a) Dataset: MovieLens-10M. RME outperformed baselines in all three groups (*p-value* ¡ 0.05).



(b) Dataset: MovieLens-20M. RME outperformed baselines in *cold-start* and *highly-active* user groups (*p-value* ¡ 0.05).



(c) Dataset: TasteProfile. RME outperformed baselines in *highly-active* user group (*p-value* ¡ 0.05).

Figure 4.7: Performance of models for three user groups. Non-directional two-sample t-test was performed. * indicates significant (*p-value* < 0.05), and *ns* indicates not significant. The error bars are the average of standard errors in the 5 folds.

4.4%, 0.9%, MAP@5 by 17.3%, 5.1%, 1.1% in *cold-start users*, *warm-start users* and *highly-active users*, respectively. Specially, in both MovieLens-10M and MovieLens-20M datasets, our model on average much improved the baselines in *cold-start users* with Recall@5, NDCG@5 and MAP@5 by 27.9%, 24.8% and 23.3%, respectively. It shows the benefit of incorporating disliked item embeddings and user embeddings. In TasteProfile dataset (Figure 4.7(c)), our model significantly improved baselines in *highly-active users* group, improving Recall@5 by 6.8%, NDCG@5 by 7.4%, and MAP@5 by 10.0% comparing to the best state-of-the-art method, while improving Recall@5 by 5.0%, NDCG@5 by 4.9%, and MAP@5 by 5.7% comparing to its best variant. However, in *cold-start users* and *warm-start users* group, RME got an equal performance comparing with the baselines (i.e., the difference between our model and other methods are not significant).

Joint learning vs separate learning. What if we conduct learning separately for each part of our model? Will the separate learning model perform better than our joint learning model? To answer the questions, we built a separate learning model as follows: first, we learned latent representations of items by jointly decomposing two SPPMI matrices $X^{(+)}$ and $X^{(-)}$ of liked item-item co-occurrences and disliked item-item co-occurrences, respectively. Then, we learned user's latent representations by minimizing the objective function in Equation (4.4), where the latent representations of items and item contexts were already learned and fixed. Next, we compared our joint learning model (i.e., RME) with the separate learning model in MovieLens-10M, MovieLens-20M, and TasteProfile datasets. Our experimental results show that our joint learning model outperformed the separate learning model by significantly improving Recall@5, NDCG@5 and MAP@5 at least 12.1%, 13.5% and 17.1%, respectively (p-value < 0.001).

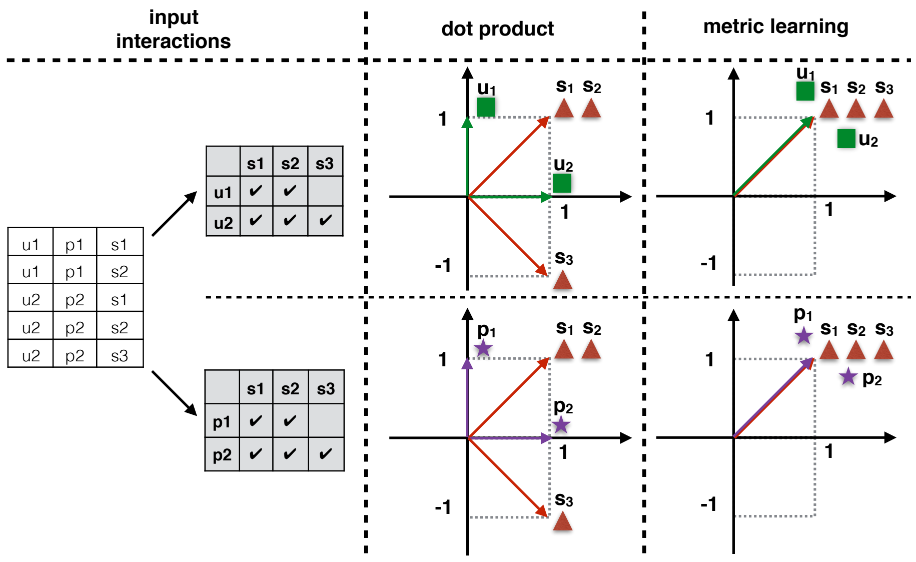


Figure 4.8: Learning with dot product vs. metric learning.

4.3 Recommending Products with a Metric Learning Based Approach

A common drawback of existing recommender systems is that they rely on the dot product to measure similarity. However, dot product is not a metric learning, so it does not convey the crucial inequality property [92, 93], and does not handle differently scaled input variables well. We illustrate the drawback of dot product in a toy example using a simple consumer-playlist-song interaction data in the automatic playlist continuation recommendation problem, and present Figure 4.8[1], where the latent dimension is size $d = 2$. Assume we have two users u_1, u_2, two playlists p_1, p_2, and three songs s_1, s_2, s_3. We can see that p_1 and p_2 (or u_1 and u_2) are similar (i.e., both liked s_1, and s_2), suggesting that s_3 would be relevant to the playlist p_1. Learning with dot product can lead to the following result: $p_1 = (0, 1), p_2 = (1, 0), s_1 = (1, 1), s_2 = (1, 1), s_3 = (1, -1)$, because $p_1^T s_1 = 1$, $p_1^T s_2 = 1$, $p_2^T s_1 = 1$, $p_2^T s_2 = 1$, $p_2^T s_3$=1 (same for users u_1, u_2). However, the dot product between p_1 and s_3 is -1, so s_3 would not be recommended to p_1. In contrast, if

[1]This Figure is inspired by [92]

we use metric learning, it will pull similar users/playlists/songs closer together by using the inequality property. In the example, the distance between p_1 and s_3 is rescaled to 0, and s_3 is now correctly portrayed as a good fit for p_1.

There exist several works that adopt metric learning for recommendation. [92] proposed *Collaborative Metric Learning* (CML) which used Euclidean distance to pull positive items closer to a user and push negative items further away. [39, 41, 94] also used Euclidean distance but for modeling transitional patterns. However, these metric-based models still fall into either *Group 1* or *Group 3*, inheriting the limitations that we described previously. Furthermore, as Euclidean distance is the primary metric, these models are highly sensitive to the scales of (latent) dimensions/variables.

According to the literature, Mahalanobis distance[1] [95, 96] overcomes the drawback (i.e., high sensitivity) of Euclidean distance. However, Mahalanobis distance has not yet been applied to recommendation with neural network designs.

Hence, in this section, we utilize Mahalanobis distance in our three novel metric learning based neural recommenders. Our first approach, *Mahalanobis Distance Based Recommender* (MDR), measures the explicit preference of a target consumer on a target item via calculating a Mahalanobis distance between the target consumer and the target item. Our second approach *Mahalanobis distance-based Attentive Item Similarity recommender* uses Mahalanobis distance to measure similarities between a target item and consumed items of the target customer. It incorporates our proposed *memory metric-based attention mechanism* that assigns attentive weights to each distance score between the target item and each member items in order to capture different influence levels. Then, our third approach fuses both two previous approaches to merge their capabilities. In addition, we incorporate customized *Adversarial Personalized Ranking* [97] into our three models to further improve their robustness.

[1]https://en.wikipedia.org/wiki/Mahalanobis_distance

In the following sections, we describe our proposed metric learning based neural recommenders in the automatic playlist continuation problem, which is the most important one in streaming platforms like Youtube, Spotify, *etc*. Note that, our methods can be applied into other problems in a similar manner. First, we will briefly mention the problem definition and preliminary about Mahalanobis distance. Then, we describe our proposed methods, and present the experiments settings and experimental results.

4.3.1 Problem Definition

Let $U = \{u_1, u_2, u_3, ..., u_m\}$ denote the set of all users, P = $\{p_1, p_2, p_3, ..., p_n\}$ denote the set of all playlists, S = $\{s_1, s_2, s_3, ..., s_v\}$ denote the set of all songs. Bolded versions of these variables, which we will introduce in the following sections, denote their respective embeddings. *m, n, v* are the number of users, playlists, and songs in a dataset, respectively. Each user $u_i \in U$ has created a set of playlists $T^{(u_i)}$ =$\{p_1, p_2, ..., p_{|T^{(u_i)}|}\}$, where each playlist $p_j \in T^{(u_i)}$ contains a list of songs $T^{(p_j)}$ =$\{s_1, s_2, ..., s_{|T^{(p_j)}|}\}$. Note that $T^{(u_1)} \cup T^{(u_2)} \cup ... \cup T^{(u_m)}$ = $\{p_1, p_2, p_3, ..., p_n\}$, $T^{(p_1)} \cup T^{(p_2)} \cup ... \cup T^{(p_n)}$ = $\{s_1, s_2, s_3, ..., s_v\}$, and the song order within each playlist is often not available in the dataset. The *Automatic Playlist Continuity (APC)* problem can then be defined as recommending new songs $s_k \notin T^{(p_j)}$ for each playlist $p_j \in T^{(u_i)}$ created by user u_i.

4.3.2 Preliminary

Given two points $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$, the Mahalanobis distance between x and y is defined as:

$$d_M(x, y) = \|x - y\|_M = \sqrt{(x - y)^T M (x - y)} \tag{4.10}$$

where $M \in \mathbb{R}^{d \times d}$ parameterizes the Mahalanobis distance metric to be learned during model training. To ensure that Eq. (4.10) produces a mathematical metric[1], M must be symmetric positive semi-definite ($M \succeq 0$). This constraint on M makes the model training process more complicated, so to ease this condition, we rewrite $M = A^T A$ ($A \in \mathbb{R}^{d \times d}$) since $M \succeq 0$. The Mahalanobis distance between two points $d_M(x, y)$ now becomes:

$$\begin{aligned} d_M(x, y) = \|x - y\|_A &= \sqrt{(x-y)^T A^T A (x-y)} \\ &= \sqrt{\big(A(x-y)\big)^T \big(A(x-y)\big)} \\ &= \|A(x-y)\|_2 = \|Ax - Ay\|_2 \end{aligned} \tag{4.11}$$

where $\| \cdot \|_2$ refers to the Euclidean distance. By rewriting Eq. (4.10) into Eq. (4.11), the Mahalanobis distance can now be computed by measuring the Euclidean distance between two linearly transformed points $x \to Ax$ and $y \to Ay$. This transformed space encourages the model to learn a more accurate similarity between x and y. $d_M(x, y)$ is generalized to basic Euclidean distance $d(x, y)$ when A is the identity matrix. If A in Eq. (4.11) is a diagonal matrix, the objective becomes learning metric A such that different dimensions are assigned different weights. Our experiments show that learning diagonal matrix A generalizes well and produces slightly better performance than if A were a full matrix. Therefore in this chapter we focus on only the diagonal case. Also note that when A is diagonal, we can rewrite Eq. (4.11) as:

$$d_M(x, y) = \|A(x-y)\|_2 = \|diag(A) \odot (x-y)\|_2 \tag{4.12}$$

where $diag(A) \in \mathbb{R}^n$ returns the diagonal of matrix A, and $\odot$ denotes the element-wise product. Therefore, we can parameterize $B = diag(A) \in \mathbb{R}^n$ and learn the Mahalanobis distance by simply computing $\|B \odot (x-y)\|_2$.

In our models' calculations, we will adopt squared Mahalanobis distance, since quadratic

[1]https://en.wikipedia.org/wiki/Metric_(mathematics)

form promotes faster learning.

4.3.3 Method

In this section, we delve into design elements and parameter estimation of our three proposed models: *Mahalanobis Distance based Recommender (MDR)*, *Mahalanobis distance-based Attentive Song Similarity recommender (MASS)*, and the combined model *Mahalanobis distance based Attentive Song Recommender (MASR)*.

Mahalanobis Distance based Recommender (MDR) *MDR* takes a target user, a target playlist, and a target song as inputs, and outputs a distance score reflecting the direct relevance of the target song to the target user's music taste and to the target playlist's theme. We will first describe how to measure each of the conditional probabilities – *P*($s_k|u_i$), *P*($s_k|p_j$), and finally *P*($s_k|u_i, p_j$) – using Mahalanobis distance. Then we will go over *MDR*'s design.

Measuring P($s_k|u_i$) Given a target user u_i, a target playlist p_j, a target song s_k, and the Mahalanobis distance $d_M(u_i, s_k)$ between u_i and s_k, *P*($s_k|u_i$) is measured by:

$$P(s_k|u_i) = \frac{\exp(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}}))}{\sum_l \exp(-(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}}))} \tag{4.13}$$

where $\beta_{\boldsymbol{s_k}}$, $\beta_{\boldsymbol{s_l}}$ are bias terms to capture their respective song's overall popularity [98]. User bias is not included in Eq.(4.13) because it is independent of *P*($s_k|u_i$) when varying candidate song s_k. The denominator $\sum_l \exp(-d_M(\boldsymbol{u_i}, \boldsymbol{s_l}) + \boldsymbol{\beta_{s_l}})$ is a normalization term shared among all candidate songs. Thus, *P*($s_k|u_i$) is measured as:

$$P(s_k|u_i) \propto -\big(d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}) + \boldsymbol{\beta_{s_k}}\big) \tag{4.14}$$

Note that training with Bayesian Personalized Ranking (BPR) will only require calculating Eq. (4.14), since for every pair of observed song k^+ and unobserved song k^- , we model the pairwise ranking $P(s_{k^+}|u_i) > P(s_{k^-}|u_i)$. Using Eq. (4.13), this inequality is satisfied only if $d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}^+) + \beta_{\boldsymbol{s_k}^+} < d_M^2(\boldsymbol{u_i}, \boldsymbol{s_k}^-) + \beta_{\boldsymbol{s_k}^-}$, which leads to Eq. (4.14).

Measuring P($s_k|p_j$) Given a target playlist p_j, a target song s_k, and the Mahalanobis distance $d_M(p_j, s_k)$ between p_j and s_k, P($s_k|p_j$) is measured by:

$$P(s_k|p_j) = \frac{\exp(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma_{s_k}}))}{\sum_l \exp(-(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_l}) + \boldsymbol{\gamma_{s_l}}))} \tag{4.15}$$

where γ_{s_k} and γ_{s_l} are song bias terms. Similar to $P(s_k|u_i)$, we shortly measure $P(s_k|p_j)$ by:

$$P(s_k|p_j) \propto -(d_M^2(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\gamma_{s_k}}) \tag{4.16}$$

Measuring P($s_k|u_i, p_j$) $P(s_k|u_i, p_j)$ is computed using the Bayesian rule under the assumption that u_i and p_j are conditionally independent given s_k:

$$\begin{aligned} P(s_k|u_i, p_j) &\propto P(u_i|s_k)P(p_j|s_k)P(s_k) \\ &= \frac{P(s_k|u_i)P(u_i)}{P(s_k)} \frac{P(s_k|p_j)P(p_j)}{P(s_k)} P(s_k) \\ &\propto P(s_k|u_i)P(s_k|p_j)\frac{1}{P(s_k)} \end{aligned} \tag{4.17}$$

In Eq. (4.17), $P(s_k)$ represents the popularity of target song s_k among the song pool. For simplicity in this chapter, we assume that selecting a random candidate song follows a uniform distribution instead of modeling this popularity information. $P(s_k|u_i, p_j)$ then becomes proportional to: $P(s_k|u_i, p_j) \propto P(s_k|u_i)P(s_k|p_j)$. Using Eq. (4.13, 4.15), we can approximate $P(s_k|u_i, p_j)$ as follows:

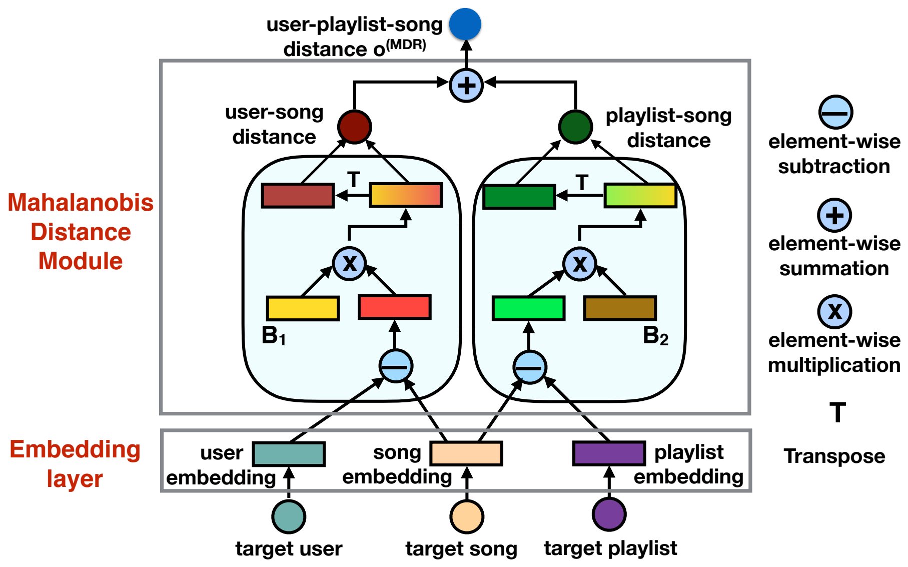


Figure 4.9: Architecture of our MDR.

$$\begin{aligned}&P(s_k|u_i,p_j) \propto \\ &\frac{\exp\big(-(d_M^2(\boldsymbol{u_i},\boldsymbol{s_k})+\boldsymbol{\beta_{s_k}})\big)}{\sum_l \exp\big(-(d_M^2(\boldsymbol{u_i},\boldsymbol{s_l})+\boldsymbol{\beta_{s_l}})\big)} \times \frac{\exp\big(-(d_M^2(\boldsymbol{p_j},\boldsymbol{s_k})+\boldsymbol{\gamma s_k})\big)}{\sum_l \exp\big(-(d_M^2(\boldsymbol{p_j},\boldsymbol{s_l})+\boldsymbol{\gamma s_l})\big)} \\ =&\frac{\exp\big(-(d_M^2(\boldsymbol{u_i},\boldsymbol{s_k})+\boldsymbol{\beta_{s_k}})-(d_M^2(\boldsymbol{p_j},\boldsymbol{s_k})+\boldsymbol{\gamma_{s_k}})\big)}{\sum_l \exp\big(-(d_M^2(\boldsymbol{u_i},\boldsymbol{s_l})+\boldsymbol{\beta_{s_l}})\big)\sum_{l'} \exp\big(-(d_M^2(\boldsymbol{p_j},\boldsymbol{s_{l'}})+\boldsymbol{\gamma_{s_{l'}}})\big)}\end{aligned} \tag{4.18}$$

Since the denominator of Eq. (4.18) is shared by all candidate songs (i.e., normalization term), we can shortly measure $P(s_k|u_i,p_j)$ by:

$$\begin{aligned}P(s_k|u_i,p_j) &\propto -\big(d_M^2(\boldsymbol{u_i},\boldsymbol{s_k})+d_M^2(\boldsymbol{p_j},\boldsymbol{s_k})\big)-\big(\boldsymbol{\beta_{s_k}}+\boldsymbol{\gamma_{s_k}}\big) \\ &= -\big(d_M^2(\boldsymbol{u_i},\boldsymbol{s_k})+d_M^2(\boldsymbol{p_j},\boldsymbol{s_k})+\boldsymbol{\theta_{s_k}}\big)\end{aligned} \tag{4.19}$$

With $P(s_k|u_i,p_j)$ now established in Eq. (4.19), we can move on to our *MDR* model.

MDR Design The *MDR* architecture is depicted in Figure 4.9. It has an Input, Embedding Layer, and Mahalanobis Distance Module.

Input: *MDR* takes a target user u_i (user ID), a target playlist p_j (playlist ID), and a target song s_k (song ID) as input.

Embedding Layer: *MDR* maintains three embedding matrices of users, playlists, and songs. By passing user u_i, playlist p_j, and song s_k through the embedding layer, we obtain their respective embedding vectors $\boldsymbol{u_i} \in \mathbb{R}^d$, $\boldsymbol{p_j} \in \mathbb{R}^d$, and $\boldsymbol{s_k} \in \mathbb{R}^d$, where d is the embedding size.

Mahalanobis Distance Module: As depicted in Figure 4.9, this module outputs a distance score $o^{(MDR)}$ that indicates the relevance of candidate song s_k to both user u_i's music preference and playlist p_j's theme. Intuitively, the lower the distance score is, the more relevant the song is. $o^{(MDR)}(\boldsymbol{u_i}, \boldsymbol{p_j}, \boldsymbol{s_k})$ is computed as follows:

$$o^{(MDR)} = o(\boldsymbol{u_i}, \boldsymbol{s_k}) + o(\boldsymbol{p_j}, \boldsymbol{s_k}) + \boldsymbol{\theta_{s_k}} \tag{4.20}$$

where $\boldsymbol{\theta_{s_k}}$ is song s_k's bias, and $o(\boldsymbol{u_i}, \boldsymbol{s_k}), o(\boldsymbol{p_j}, \boldsymbol{s_k})$ are quadratic Mahalanobis distance scores between user u_i and song s_k, and between playlist p_j and song s_k, shown in the following two equations. $\boldsymbol{B_1} \in \mathbb{R}^d$ and $\boldsymbol{B_2} \in \mathbb{R}^d$ are two metric learning vectors. And,

$$\begin{aligned} o(\boldsymbol{u_i}, \boldsymbol{s_k}) &= \big(\boldsymbol{B_1} \odot (\boldsymbol{u_i} - \boldsymbol{s_k})\big)^T \big(\boldsymbol{B_1} \odot (\boldsymbol{u_i} - \boldsymbol{s_k})\big) \\ o(\boldsymbol{p_j}, \boldsymbol{s_k}) &= \big(\boldsymbol{B_2} \odot (\boldsymbol{p_j} - \boldsymbol{s_k})\big)^T \big(\boldsymbol{B_2} \odot (\boldsymbol{p_j} - \boldsymbol{s_k})\big) \end{aligned}$$

Mahalanobis distance-based Attentive Song Similarity recommender (MASS) An overview of *MASS*'s architecture is depicted in Figure 4.10. *MASS* has five components: Input, Embedding Layer, Processing Layer, Attention Layer, and Output.

Input: The inputs to our *MASS* model include a target user u_i, a candidate song s_k for a target playlist p_j, and a list of l member songs within the playlist, where l is the number of songs in the largest playlist (i.e., containing the largest number of songs) in the dataset. If a playlist contains less than l songs, we pad the list with *zeroes* until it reaches length l.

Embedding Layer: This layer holds two embedding matrices: a user embedding matrix $\mathbf{U} \in \mathbb{R}^{m \times d}$ and a song embedding matrix $\mathbf{S} \in \mathbb{R}^{v \times d}$. By passing the input target user u_i and target song s_k through these two respective matrices, we obtain their embedding vectors $\boldsymbol{u_i} \in \mathbb{R}^d$ and $\boldsymbol{s_k} \in \mathbb{R}^d$. Similarly, we acquire the embedding vectors for all l

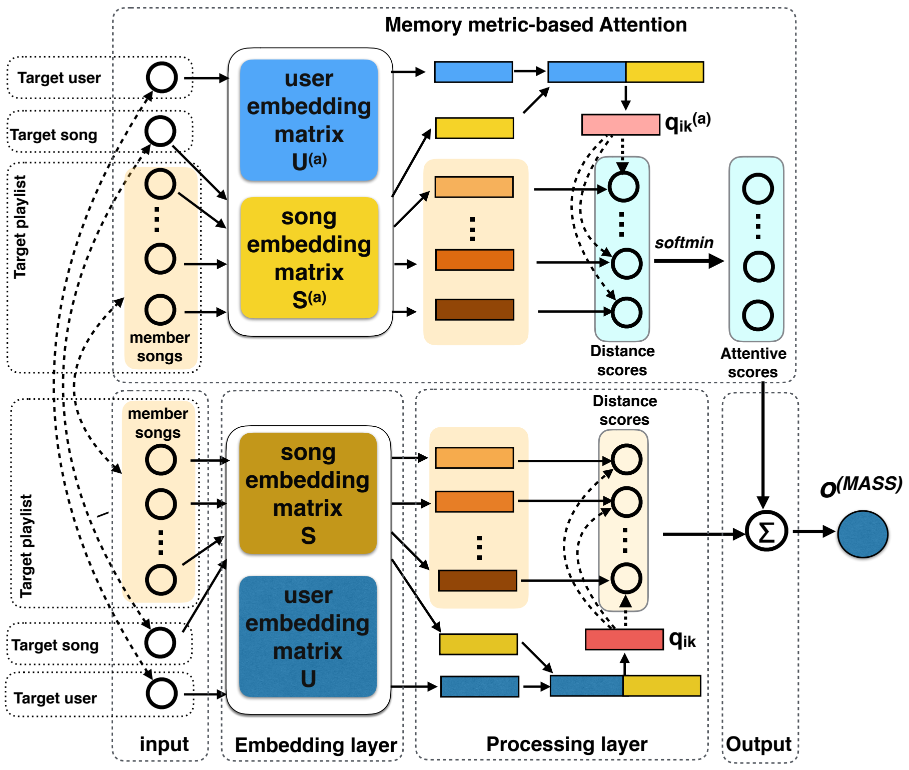


Figure 4.10: Architecture of our MASS.

member songs in p_j, denoted by $\boldsymbol{s_1}, \boldsymbol{s_2}, ..., \boldsymbol{s_l}$.

Processing Layer: We first need to consolidate $\boldsymbol{u_i}$ and $\boldsymbol{s_k}$. Following widely adopted deep multimodal network designs [99], we concatenate the two embeddings, and then transform them into a new vector $\boldsymbol{q}_{ik} \in \mathbb{R}^d$ via a fully connected layer with weight matrix $\boldsymbol{W}_1 \in \mathbb{R}^{2d \times d}$, bias term $\boldsymbol{b} \in \mathbb{R}$, and activation function $\texttt{ReLU}$. We formulate this process as follows:

$$\boldsymbol{q}_{ik} = \texttt{ReLU}\left(\mathrm{W}_1 \begin{bmatrix} \boldsymbol{u}_i \\ \boldsymbol{s}_k \end{bmatrix} + \boldsymbol{b}_1\right) \tag{4.21}$$

Note that $\boldsymbol{q}_{ik}$ can be interpreted as a search query in QA systems [100, 101]. Since we combined the target user u_i with the query song s_k (to add to the user's target playlist), the search query $\boldsymbol{q}_{ik}$ is personalized. The $\texttt{ReLU}$ activation function models a non-linear combination between these two target entities, and was chosen over *sigmoid* or *tanh* due to its encouragement of sparse activations and proven non-saturation [102], which helps prevent overfitting.

Next, given the embedding vectors $\boldsymbol{s_1}, \boldsymbol{s_2}, ..., \boldsymbol{s_l}$ of the l member songs in target playlist p_j, we approximate the conditional probability $P(s_k|u_i, s_1, s_2, ..., s_l)$ by:

$$P(s_k|u_i, s_1, s_2, ..., s_l) \propto -\Bigg(\sum_{t=1}^{l} \boldsymbol{\alpha_{ikt}} d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) + \boldsymbol{b_{s_k}}\Bigg) \qquad (4.22)$$

where $d_M(\cdot)$ returns the Mahalanobis distance between two vectors, b_{s_k} is the song bias reflecting its overall popularity, and $\boldsymbol{\alpha}_{ikt}$ is the attention score to weight the contribution of the partial distance between search query $\boldsymbol{q_{ik}}$ and member song $\boldsymbol{s_t}$. We will show how to calculate $d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t})$ below, and $\boldsymbol{\alpha_{ikt}}$ in *Attention Layer* at 4.3.3.

As indicated in Eq. (4.12), we parameterize $\boldsymbol{B_3} \in \mathbb{R}^d$, which will be learned during the training phase. The Mahalanobis distance between the search query $\boldsymbol{q_{ik}}$ and each member song $\boldsymbol{s_t}$, treating $\boldsymbol{B_3}$ as an edge-weight vector, is measured by:

$$d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) = \left\|\boldsymbol{e_{ikt}^T e_{ikt}}\right\|_2^2 \quad \text{where} \quad \boldsymbol{e_{ikt}} = \boldsymbol{B_3} \odot (\boldsymbol{q_{ik}} - \boldsymbol{s_t}) \qquad (4.23)$$

Calculating Eq. (4.23) for every member song $\boldsymbol{s_t}$ yields the following l-dimensional vector:

$$\begin{bmatrix} d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_1) \\ d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_2) \\ \cdots \\ d_M^2(\boldsymbol{q}_{ik}, \boldsymbol{s}_l) \end{bmatrix} = \begin{bmatrix} \left\|\boldsymbol{e}_{ik1}^T \boldsymbol{e}_{ik1}\right\|_2^2 \\ \left\|\boldsymbol{e}_{ik2}^T \boldsymbol{e}_{ik2}\right\|_2^2 \\ \cdots \\ \left\|\boldsymbol{e}_{ikl}^T \boldsymbol{e}_{ikl}\right\|_2^2 \end{bmatrix} \qquad (4.24)$$

Note that $\boldsymbol{B_3}$ is shared across all Mahalanobis measurement pairs. Now we go into detail of how to calculate the attention weights $\boldsymbol{\alpha_{ikt}}$ using our proposed Attention Layer.

Attention Layer: With l distance scores obtained in Eq. (4.24), we need to combine them into one distance value to reflect how relevant the target song is *w.r.t* the target playlist's member songs. The simplest approach is to follow the well-known item similarity design [36, 103] where the same weights are assigned for all l distance scores. This is sub-optimal in our domain because different member song can relate to the target song differently. For example, given a country playlist and a target song of the same genre, the member songs that share the same artist with the target song would be more similar to the target song than the other member songs in the playlist. To address this concern, we propose a novel *memory metric-based attention mechanism* to properly allocate different attentive scores to the distance values in Eq. (4.24). Compared to existing attention mechanisms, our attention mechanism maintains its own embedding memory of users and songs (i.e., memory-based property), which can function as an external memory. It also computes attentive scores using Mahalanobis distance (i.e., metric-based property) instead of traditional dot product. Note that the memory-based property is also commonly applied to question-answering in NLP, where memory networks have utilized external memory [104] for better memorization of context information [105, 106]. Our attention mechanism has one external memory containing user and song embedding matrices. When the user and song embedding matrices of our attention mechanism are identical to those in the embedding layer, it is the same as looking up the embedding vectors of target users, target songs, and member songs in the embedding layer (Section 4.3.3). Therefore, using external memory will make room for more flexibility in our models.

The attention layer features an external user embedding matrix $\mathbf{U}^{(\mathbf{a})} \in \mathbb{R}^{m \times d}$ and external song embedding matrix $\mathbf{S}^{(\mathbf{a})} \in \mathbb{R}^{v \times d}$. Given the following inputs – a target user u_i, a target song s_k, and all l member songs in playlist p_j – by passing them through the corresponding embedding matrices, we obtain the embedding vectors of u_i, s_k, and all the member songs, denoted as $\boldsymbol{u_i}^{(a)}$, $\boldsymbol{s_k}^{(a)}$, and $\boldsymbol{s_1}^{(a)}, \boldsymbol{s_2}^{(a)}, ..., \boldsymbol{s_l}^{(a)}$, respectively.

We then forge a personalized search query $\boldsymbol{q_{ik}}^{(a)}$ by combining $\boldsymbol{u_i}^{(a)}$ and $\boldsymbol{s_k}^{(a)}$ in a multimodal design as follows:

$$\boldsymbol{q}_{\boldsymbol{ik}}^{(a)} = \texttt{ReLU}\left(\mathbf{W}_2 \begin{bmatrix} \boldsymbol{u}_i^{(a)} \\ \boldsymbol{s}_k^{(a)} \end{bmatrix} + \boldsymbol{b}_2\right) \tag{4.25}$$

where $\mathbf{W}_2 \in \mathbb{R}^{2d \times d}$ is a weight matrix and $\boldsymbol{b}_2$ is a bias term. Next, we measure the Mahalanobis distance (with an edge weight vector $\boldsymbol{B_4} \in \mathbb{R}^d$) from $\boldsymbol{q}_{\boldsymbol{ik}}^{(a)}$ to a member song's embedding vector $\boldsymbol{s}_{\boldsymbol{t}}^{(a)}$ where $t \in \overline{1, l}$:

$$d_M^2(\boldsymbol{q}_{\boldsymbol{ik}}^{(a)}, \boldsymbol{s}_{\boldsymbol{t}}^{(a)}) = \left\|\left(\boldsymbol{e}_{\boldsymbol{ikt}}^{(a)}\right)^T \boldsymbol{e}_{\boldsymbol{ikt}}^{(a)}\right\|_2^2 \quad \text{where} \quad \boldsymbol{e}_{\boldsymbol{ikt}}^{(a)} = \boldsymbol{B_4} \odot \left(\boldsymbol{q}_{\boldsymbol{ik}}^{(a)} - \boldsymbol{s}_{\boldsymbol{t}}^{(\boldsymbol{a})}\right) \tag{4.26}$$

Using Eq. (4.26), we generate l distance scores between each of l member songs and the candidate song. Then we apply *softmin* on l distance scores in order to obtain the member songs' attentive scores[1]. Intuitively, the lower the distance between a search query and a member song vector, the higher its contribution level is *w.r.t* the candidate song.

$$\boldsymbol{\alpha_{ikt}} = \frac{\exp\left(-\left\|\left(\boldsymbol{e}_{\boldsymbol{ikt}}^{(a)}\right)^T \boldsymbol{e}_{\boldsymbol{ikt}}^{(a)}\right\|_2^2\right)}{\sum_{t\prime=1}^{l} \exp\left(-\left\|\left(\boldsymbol{e}_{\boldsymbol{ikt\prime}}^{(a)}\right)^T \boldsymbol{e}_{\boldsymbol{ikt\prime}}^{(a)}\right\|_2^2\right)} \tag{4.27}$$

Output: We output the total attentive distances $o^{(MASS)}$ from the target song s_k to target playlist p_j's existing songs by:

$$\mathbf{o}^{(\mathbf{MASS})} = -\left(\sum_{t=1}^{l} \boldsymbol{\alpha_{ikt}} d_M^2(\boldsymbol{q_{ik}}, \boldsymbol{s_t}) + \boldsymbol{b_{s_k}}\right) \tag{4.28}$$

where $\boldsymbol{\alpha_{ikt}}$ is the attentive score from Eq. (4.27), $d_M(\boldsymbol{q_{ik}}, \boldsymbol{s_t})$ is the personalized Mahalanobis distance between target song s_k and a member song s_t in user u_i's playlist (Eq. (4.24)), $\boldsymbol{b_{s_k}}$ is the song bias.

[1]Note that attentive scores of padded items are 0.

Mahalanobis distance based Attentive Song Recommender (MASR = MDR + MASS) We enhance our performance on the *APC* problem by combining our *MDR* and *MASS* into a *Mahalanobis distance based Attentive Song Recommender (MASR)* model. *MASR* outputs a cumulative distance score from the outputs of *MDR* and *MASS* as follows:

$$\mathbf{o}^{(\mathbf{MASR})} = \alpha \mathbf{o}^{(\mathbf{MDR})} + (1-\alpha)\mathbf{o}^{(\mathbf{MASS})} \tag{4.29}$$

where $\mathbf{o}^{(\mathbf{MDR})}$ is from Eq. (4.20), $\mathbf{o}^{(\mathbf{MASS})}$ is from Eq. (4.28), and $\alpha \in [0,1]$ is a hyper-parameter to adjust the contribution levels of *MDR* and *MASS*. α can be tuned using a development dataset. However, in the following experiments, we set $\alpha = 0.5$ to receive equal contribution from *MDR* and *MASS*. We pretrain *MDR* and *MASS* first, then fix *MDR* and *MASS*'s parameters in *MASR*. There are two benefits of this design. First, if MASR is learnable with pretrained MDR and MASS initialization, MASR would have too high a computational cost to train. Second, by making *MASR* non-learnable, *MDR* and *MASS* in *MASR* can be trained separately and in parallel, which is more practical and efficient for real-world systems.

Parameter Estimation Learning with Bayesian Personalized Ranking (BPR) loss We apply BPR loss as an objective function to train our *MDR, MASS, MASR* as follows:

$$\mathcal{L}(\mathcal{D}|\Theta) = \underset{\Theta}{\operatorname{argmin}} \Big(- \sum_{(i,j,k^+,k^-)} \log \sigma(\mathbf{o}_{ijk^-} - \mathbf{o}_{ijk^+}) + \lambda_\Theta \|\Theta\|^2 \Big) \tag{4.30}$$

where (i, j, k^+, k^-) is a quartet of a target user, a target playlist, a positive song, and a negative song which is randomly sampled. $\sigma(\cdot)$ is the *sigmoid* function; $\mathcal{D}$ denotes all training instances; Θ are the model's parameters (for instance, $\Theta = \{\mathbf{U}, \mathbf{S}, \mathbf{U^{(a)}}, \mathbf{S^{(a)}}, \mathbf{W_1}, \mathbf{W_2}, \mathbf{B_3}, \mathbf{B_4}, \mathbf{b}\}$ in the *MASS* model); λ_Θ is a regularization hyper-parameter; and $\mathbf{o_{ijk}}$ is the output of either *MDR*, *MASS*, or *MASR*, which is measured in Eq. (4.20), (4.28), and (4.29), respectively.

Learning with Adversarial Personalized Ranking (APR) loss It has been shown in [97] that BPR loss is vulnerable to adversarial noise, and *APR* was proposed to enhance the robustness of a simple matrix factorization model. In this work, we apply *APR* to further improve the robustness of our *MDR*, *MASS*, and *MASR*. We name our *MDR*, *MASS*, and *MASR* trained with *APR* loss as *AMDR*, *AMASS*, *AMASR* by adding an "adversarial (A)" term, respectively. Denote δ as adversarial noise on the model's parameters Θ. The *BPR* loss from adding adversarial noise δ to Θ is defined by:

$$\mathcal{L}(\mathcal{D}|\hat{\Theta}+\delta) = \underset{\Theta=\hat{\Theta}+\delta}{\operatorname{argmax}} \Big(- \sum_{(i,j,k^+,k^-)} \log \sigma(\mathbf{o}_{ijk^-} - \mathbf{o}_{ijk^+}) \Big) \tag{4.31}$$

where $\hat{\Theta}$ is optimized in Eq. (4.30) and fixed as constants in Eq. (4.31). Then, training with *APR* aims to play a minimax game as follows:

$$\arg\min_{\Theta} \max_{\delta, \|\delta\| \leq \epsilon s(\hat{\Theta})} \mathcal{L}(\mathcal{D}|\Theta) + \lambda_\delta \mathcal{L}(\mathcal{D}|\hat{\Theta}+\delta) \tag{4.32}$$

where ϵ is a hyper-parameter to control the magnitude of perturbations δ. In [97], the authors fixed ϵ for all the model's parameters, which is not ideal because different parameters can endure different levels of perturbation. If we add too large adversarial noise, the model's performance will downgrade, while adding too small noise does not guarantee more robust models. Hence, we multiply ϵ with the standard deviation $s(\hat{\Theta})$ of the targeting parameter $\hat{\Theta}$ to provide a more flexible noise magnitude. For instance, the adversarial noise magnitude on parameter $\mathbf{B_3}$ in *AMASS* model is $\epsilon \times s(\mathbf{B_3})$. If the values in $\mathbf{B_3}$ are widely dispersed, they are more vulnerable to attack, so the adversarial noise applied during training must be higher in order to improve robustness. Whereas if the values are centralized, they are already robust, so only a small noise magnitude is needed.

Learning with *APR* follows 4 steps: **Step 1**: unlike [97] where parameters are saved at the last training epoch, which can be over-fitted parameter values (e.g. some thousands of epoches for matrix factorization in [97]), we first learn our models' parameters by minimizing Eq. (4.30) and save the best checkpoint based on evaluating on a development dataset. **Step 2:** with optimal $\hat{\Theta}$ learned in *Step 1*, in Eq. (4.31), we set $\Theta = \hat{\Theta}$ and fix Θ to learn δ. **Step 3:** with optimal $\hat{\delta}$ learned in Eq. (4.31), in Eq. (4.32) we set $\delta = \hat{\delta}$ and fix δ to learn new values for Θ. **Step 4**: We repeat *Step 2* and *Step 3* until a maximum number of epochs is reached and save the best checkpoint based on evaluation on a development dataset. Following [97, 107], the update rule for δ is obtained by using the fast gradient method as follows:

$$\delta = \epsilon \times s(\hat{\Theta}) \times \frac{\nabla_\delta(\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta))}{\left\|\nabla_\delta(\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta))\right\|_2} \tag{4.33}$$

Note that update rules of parameters in Θ can be easily obtained by computing the partial derivative *w.r.t* each parameter in Θ.

Time Complexity

Let Ω denote the total number of training instances (= $\sum_j N(p_j)$ where $N(p_j)$ refers to the number of songs in training playlist p_j). $\omega = max(N(p_j)), \forall j = \overline{1,n}$ denotes the maximum number of songs in all playlists. For each forward pass, *MDR* takes $\mathcal{O}(d)$ to measure $\mathbf{o}^{(MDR)}$ (in Eq. (4.20)) for a positive training instance, and another forward pass with $\mathcal{O}(d)$ to calculate $\mathbf{o}^{(MDR)}$ for a negative instance. The backpropagation for updating parameters take the same complexity. Therefore, the time complexity of *MDR* is $\mathcal{O}(\Omega d)$. Similarly, for each positive training instance, *MASS* takes (i) $\mathcal{O}(2d^2)$ to make each query in Eq. (4.21) and Eq. (4.25); (ii) $\mathcal{O}(\omega d)$ to calculate ω distance scores from ω member songs to the target song in Eq. (4.24); and (iii) $\mathcal{O}(\omega d)$ to measure attention scores in Eq. (4.27). Since embedding size d is often small, $\mathcal{O}(\omega d)$ is a dominant term and *MASS*'s time complexity is $\mathcal{O}(\Omega\omega d)$. Hence, both *MDR* and *MASS* scale linearly to the number of training instances and can run very fast, especially with sparse datasets. When training with *APR*, updating δ in Eq. (4.33) with fixed $\hat{\Theta}$ needs one forward and one

Table 4.3: Statistics of datasets.

Statistics	**30Music**	**AOTM**
# of users	12,336	15,835
# of playlists	32,140	99,903
# of songs	276,142	504,283
# of interactions	666,788	1,966,795
avg. # of playlists per user	2.6	6.3
avg. & max # of songs per playlist	18.75 & 63	17.69 & 58
Density	0.008%	0.004%

backward pass. Learning Θ in Eq. (4.32) requires one forward pass to measure $\mathcal{L}(\mathcal{D}|\Theta)$ in Eq. (4.30), one forward pass to measure $\mathcal{L}(\mathcal{D}|\hat{\Theta} + \delta)$ in Eq. (4.31), and one backward pass to update Θ in Eq. (4.32). Hence, time complexity when training with *APR* is h times higher (h is small) compared to training with *BPR* loss.

4.3.4 Experimental Settings

Datasets:

To evaluate our proposed models and existing baselines, we used two publicly accessible real-world datasets that contain user, playlist, and song information. They are described as follows:

- 30Music [108]: This is a collection of playlists data retrieved from Internet radio stations through Last.fm[1]. It consists of 57K playlists and 466K songs from 15K users.
- AOTM [109]: This dataset was collected from the Art of the Mix[2] playlist database. It consists of 101K playlists and 504K songs from 16K users, spanning from Jan 1998 to June 2011.

For data preprocessing, we removed duplicate songs in playlists. Then we adopted a

[1]https://www.last.fm

[2]http://www.artofthemix.org/

widely used *k-core* preprocessing step [88, 110] (with *k-core* = 5), filtering out playlists with less than 5 songs. We also removed users with an extremely large number of playlists, and extremely large playlists (i.e., containing thousands of songs). Since the datasets did not have song order information for playlists (i.e., which song was added to a playlist first, then next, and so on), we randomly shuffled the song order of each playlist and used it in the sequential recommendation baseline models to compare with our models. The two datasets are implicit feedback datasets. The statistics of the preprocessed datasets are presented in Table 4.3.

Baselines:

We compared our proposed models with **eight** strong state-of-the-art models in the *APC* task. The baselines were trained by using *BPR* loss for a fair comparison:

- **Bayesian Personalized Ranking (MF-BPR)** [111]: It is a pairwise matrix factorization method for implicit feedback datasets.
- **Collaborative Metric Learning (CML)** [92]: It is a collaborative metric-based method. It adopted Euclidean distance to measure a user's preference on items.
- **Neural Collaborative Filtering (NeuMF++)** [26]: It is a neural network based method that models non-linear user-item interactions. We pretrained two components of NeuMF to obtain its best performance (i.e., NeuMF++).
- **Factored Item Similarity Methods (FISM)** [36]: It is a item neighborhood-based method. It ranks a candidate song based on its similarity with member songs using dot product.
- **Collaborative Memory Network (CMN++)** [29]: It is a user-neighborhood based model using a memory network to assign attentive scores for similar users.
- **Personalized Ranking Metric Embedding (PRME)** [39]: It is a sequential recommender that models a personalized first-order Markov behavior using Euclidean dis-

tance.

- **Translation-based Recommendation (Transrec)** [41]: It is one of the best sequential recommendation methods. It models the third order between the user, the previous song, and the next song where the user acts as a translator.
- **Convolutional Sequence Embedding Recommendation (Caser)** [42]: It is a CNN based sequential recommendation. It embeds a sequence of recent songs into an "image" in time and latent spaces, then learns sequential patterns as local features of the image using different horizontal and vertical filters.

We did not compare our models with baselines that performed worse than above listed baselines like *item-KNN*[90], *SLIM*[103], *etc.*

MF-BPR, CML, and NeuMF++ used only user/playlist-song interaction data to model either users' preferences over songs *P*($s|u$) or playlists' tastes over songs *P*($s|p$). We ran the baselines both ways, and report the best results. Two neighborhood-based baselines utilized neighbor users/playlists (i.e., CMN++) or member songs (i.e., FISM) to recommend the next song based on user/playlist similarities or song similarities (i.e., measure *P*($s|u, s_1, s_2, ..., s_l$) and *P*($s|p, s_1, s_2, ..., s_l$), of which we report the best results).

Protocol: We use the widely adopted *leave-one-out* evaluation setting [26]. Since both the 30Music and AOTM datasets do not contain timestamps of added songs for each playlist, we randomly sample two songs per playlist–one for a positive test sample, and one for a development set to tune hyper-parameters–while the remaining songs in each playlist make up the training set. We follow [26, 34] and uniformly random sample 100 non-member songs as negative songs, and rank the test song against those negative songs.

Evaluation metrics: We evaluate the performance of the models with two widely used metrics: Hit Ratio (*hit@N*), and Normalized Discounted Cumulative Gain (*NDCG@N*). The *hit@N* measures whether the test item is in the recommended list or not, while the *NDCG@N* takes into account the position of the *hit* and assigns higher scores to hits

Table 4.4: Performance of the baselines, and our models. The last two lines show the relative improvement of MASR and AMASR compared to the best baseline.

	Methods	**30Music**		**AOTM**	
		hit@10	ndcg@10	hit@10	ndcg@10
(a)	MF-BPR	0.450	0.315	0.699	0.473
(b)	CML	0.600	0.452	0.735	0.481
(c)	NeuMF++	0.623	0.461	0.741	0.498
(d)	FISM	0.544	0.346	0.686	0.446
(e)	CMN++	0.536	0.397	0.722	0.505
(f)	PRME	0.426	0.260	0.570	0.354
(g)	Transrec	0.570	0.417	0.710	0.450
(h)	Caser	0.458	0.289	0.681	0.448
Ours	MDR	0.705	0.524	0.820	0.631
	MASS	0.670	0.500	0.834	0.639
	MASR	**0.731**	**0.564**	**0.854**	**0.654**
	AMDR	0.764	0.581	0.850	0.658
	AMASS	0.753	0.581	0.856	0.659
	AMASR	**0.785**	**0.604**	**0.874**	**0.677**
Imprv. (%)	MASR	**+17.34**	**+22.34**	**+13.36**	**+28.24**
	AMASR	**+26.00**	**+31.02**	**+17.95**	**+34.19**

at top-rank positions. For the test set, we measure both metrics and report the average scores.

Hyper-parameters settings: Models are trained with the *Adam* optimizer with learning rates from $\{0.001, 0.0001\}$, regularization terms λ_Θ from $\{0, 0.1, 0.01, 0.001, 0.0001\}$, and embedding sizes from $\{8, 16, 32, 64\}$. The maximum number of epochs is 50, and the batch size is 256. The number of hops in *CMN*++ are selected from $\{1, 2, 3, 4\}$. In *NeuMF*++, the number of MLP layers are selected from $\{1, 2, 3\}$. The number of negative samples per one positive instance is 4, similar to [26]. The Markov order L in *Caser* is selected from $\{4, 5, 6, 7, 8, 9, 10\}$. For *APR* training, the number of *APR* training epochs is 50, the noise magnitude ϵ is selected from $\{0.5, 1.0\}$, and the adversarial regularization λ_δ is set to 1, as suggested in [97]. Adversarial noise is added only in training process, and are initialized as *zero*. All hyper-parameters are tuned by using the

Table 4.5: Performance of variants of our MDR and MASS. RI indicates relative average improvement over the corresponding method.

Methods	30Music		AOTM		RI(%)
	hit@10	ndcg@10	hit@10	ndcg@10	
MDR_us	0.684	0.500	0.815	0.594	**+3.68**
MDR_ps	0.654	0.476	0.746	0.547	**+10.79**
MDR_ups (i.e., MDR)	**0.705**	**0.524**	**0.818**	**0.613**	
MASS_ups	0.651	0.479	0.789	0.581	**+4.12**
MASS_ps	0.621	0.450	0.764	0.523	**+10.82**
MASS_us (i.e., MASS)	**0.670**	**0.500**	**0.820**	**0.631**	

development set. Our source code is available at *https://github.com/thanhdtran/MASR.git*.

4.3.5 Experimental Results

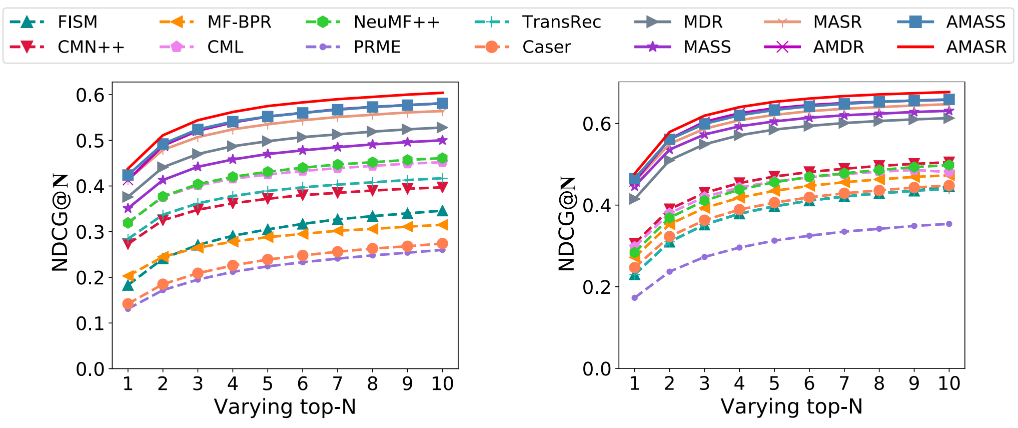
Performance comparison: Table 4.4 shows the performance of our proposed models and baselines on each dataset. *MDR* and baselines (a)-(c) are in *Group 1*, but *MDR* shows much better performance compared to the (a)-(c) baselines, improving at least 11.14% *hit@10* and 18.81% *NDCG@10* on average. *CML* simply adopts Euclidean distance between users/playlists and positive songs, but has nearly equal performance with NeuMF++, which utilizes a neural network to learn non-linear relationships between users/playlists and songs. This result shows the effectiveness of using metric learning over dot product in recommendation. *MDR* outperforms *CML* by 19.04% on average. This confirms the effectiveness of Mahalanobis distance over Euclidian distance for recommendation.

MASS outperforms both *FISM* and *CMN++*, improving *hit@10* by 18.4%, and *NDCG@10* by 25.5% on average. This is because *FISM* does not consider the attentive contribution of different neighbors. Even though *CMN++* can assign attention scores for different user/playlist neighbors, it bears the flaws of *Group 1* by considering only either neighbor users or neighbor playlists. More importantly, *MASS* uses a novel attentive metric design, while dot product is utilized in *FISM* and *CMN*++. Sequential models, (f)-(h) baselines, do not work well. In particular, *MASS* outperforms the (f)-(h) baselines, improving 24.6% on average compared to the best model in (f)-(h).

MASR outperforms both *MDR* and *MASS*, indicating the effectiveness of fusing them into one model. Particularly, *MASR* improves *MDR* by 5.0%, and *MASS* by 6.7% on average. Performances of *MDR, MASS, MASR* are boosted when adopting *APR* loss with a *flexible* noise magnitude. *AMDR* improves *MDR* by 7.7%, *AMASS* improves *MASS* by 9.4%, and *AMASR* improves *MASR* by 5.8%. We also compare our *flexible* noise magnitude with a fixed noise magnitude used in [97] by varying the fixed noise magnitude in {0.5, 1.0} and setting $\lambda_\delta = 1$. We observe that *APR* with a *flexible* noise magnitude performs better with an average improvement of 7.53%.

Next, we build variants of our *MDR* and *MASS* models by removing either playlist or user embeddings, or using both of them. Table 4.5 presents an ablation study of exploiting playlist embeddings. *MDR_us* is the *MDR* that uses only user-song interactions (i.e., ignore playlist-song distance $o(p_j, s_k)$ in Eq. (4.20)). *MDR_ps* is the *MDR* that uses only playlist-song interactions (i.e., ignores user-song distance $o(u_i, s_k)$ in Eq. (4.20)). *MDR_ups* is our proposed *MDR* model. Similarly, *MASS_ups* is the *MASS* model but considers both user-song distances and playlist-song distances in its design. The *Embedding Layer* and *Attention Layer* of *MASS_ups* have additional playlist embedding matrices $\mathbf{P} \in \mathbb{R}^{n\times d}$ and $\mathbf{P^{(a)}} \in \mathbb{R}^{n\times d}$, respectively. *MASS_ps* is the *MASS* model that replaces user embeddings with playlist embeddings. *MASS_us* is our proposed *MASS* model.

MDR (i.e., *MDR_ups*) outperforms its derived forms (*MDR_us* and *MDR_ps*), improving by 3.7$\sim$10.8% on average. This result shows the effectiveness of modeling both users' preferences and playlists' themes in *MDR* design. *MASS* (i.e., *MASS_us*) outperforms its two variants (*MASS_ups* and *MASS_ps*), improving *MASS_ups* by 3.7%, and *MASS_ps* by



(a) 30Music. (b) AOTM.

Figure 4.11: Performance of our models and the baselines when varying *N* (or *top-N* recommendation list) from [1, 10].

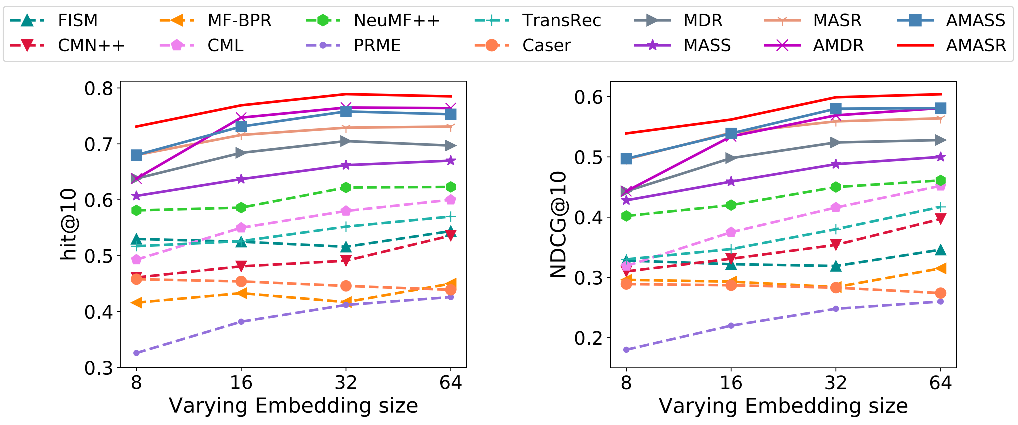


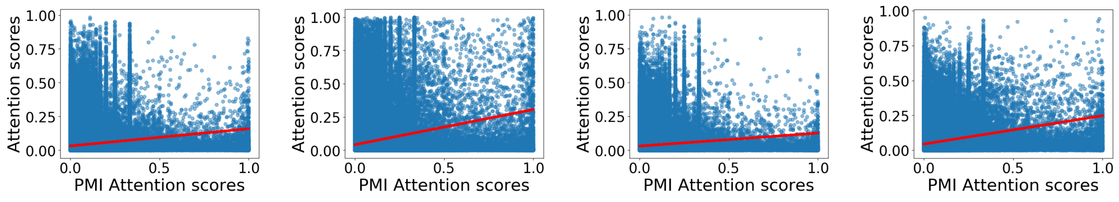
Figure 4.12: Performance of all models when varying the embedding size *d* from {8, 16, 32, 64} in 30Music dataset.

10.8% on average. It makes sense that using additional playlist embeddings in *MASS_ups* is redundant since the member songs have already conveyed the playlist's theme, and ignoring user embeddings in *MASS_ps* neglects user preferences.

Varying top-N recommendation list and embedding size: Figure 4.11 shows performances of all models when varying *top-N* recommendation from 1 to 10. We see that all models gain higher results when increasing *top-N*, and all our proposed models outperform all baselines across all *top-N* values. On average, *MASR* improves 26.3%, and *AMASR* improves 33.9% over the best baseline's performance.

Table 4.6: Performance of MASS using various attention mechanisms.

Attention Types	30Music		AOTM		RI(%)
	hit@10	ndcg@10	hit@10	ndcg@10	
non-mem + dot	0.630	0.454	0.785	0.574	**+8.51**
non-mem + metric	0.660	0.490	0.803	0.601	**+3.43**
mem + dot	0.659	0.475	0.791	0.585	**+5.40**
mem + metric	**0.670**	**0.500**	**0.834**	**0.639**	



(a) ρ=0.153 (b) ρ=0.215 (c) ρ=0.171 (d) ρ=0.254

Figure 4.13: Scatter plots of PMI attention scores vs. attention weights learned by various attention mechanisms, showing corresponding Pearson correlation score ρ). (a)non-mem + dot, (b)non-mem + metric, (c)mem + dot, (d)mem + metric.

Figure 4.12[1] shows all models' performances when varying the embedding size d from $\{8, 16, 32, 64\}$ for the *30Music* dataset. Note that the *AOTM* dataset also shows similar results but is omitted due to the space limitations. We observe that most models tend to have increased performance when increasing embedding size. *AMDR* does not improve *MDR* when $d = 8$ but does so when increasing d. This phenomenon was also reported in [97] because when $d = 8$, *MDR* is too simple and has a small number of parameters, which is far from overfitting the data and not very vulnerable to adversarial noise. However, for more complicated models like *MASS* and *MASR*, even with a small embedding size $d = 8$, *APR* shows its effectiveness in making the models more robust, and leads to an improvement of *AMASS* by 12.0% over *MASS*, and an improvement of *AMASR* by 7.5% over *MASR*. The improvements of *AMDR, AMASS, AMASR* over their corresponding base models are higher for larger d due to the increase of model complexity.

[1]Figure 4.12 shares the same legend with Figure 4.11 for saving space.

Is our memory metric-based attention helpful? To answer this question, we evaluate how *MASS*'s performance changed when varying its attention mechanism as follows:

- *non-memory* + *dot product* (*non-mem* + *dot*): It is the popular *dot attention* introduced in [112].
- *non-memory* + *metric* (*non-mem* + *metric*): It is our proposed attention with Mahalanobis distance but no external memory.
- *memory* + *dot product* (*mem* + *dot*): It is the *dot attention* but exploiting external memory.
- *memory* + *metric* (*mem* + *metric*): It is our proposed attention mechanism.

We do not compare with the *no-attention* case because literature has already proved the effectiveness of the attention mechanism [113]. Table 4.6 shows the performance of *MASS* under the variations of our proposed attention mechanism. We have some key observations. First, *non-mem* + *metric* attention outperforms *non-mem* + *dot* attention with an improvement of 4.9% on average. Similarly, *mem* + *metric* attention improves the *mem* + *dot* attention design by 5.4% on average. This enhancement comes from different nature of metric space and dot product space. Moreover, these results confirm that metric-based attention designs fit better into our proposed Mahalanobis distance based model. Second, *memory* based attention works better than *non-mem* attention. Particularly, on average, *mem* + *dot* improves *non-mem* + *dot* by 2.98%, and *mem* + *metric* improves *non-mem* + *metric* by 3.43%. Overall, the performance order is *mem* + *metric* ¿ *non-mem* + *metric* ¿ *mem* + *dot* ¿ *non-mem* + *dot*, which confirms that our proposed attention performs the best and improves 3.43$\sim$8.51% compared to its variations.

Deep analysis on attention: To further understand how attention mechanisms work, we connect attentive scores generated by attention mechanisms with *Pointwise Mutual*

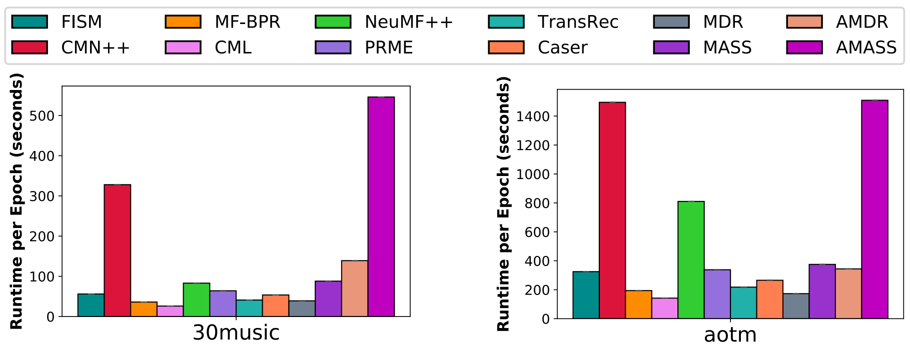


Figure 4.14: Runtime of all models in 30Music and AOTM.

Information scores. Given a target song k and a member song t, the *PMI* score between them is defined as: $PMI(k,t) = log\frac{P(k,t)}{P(k)\times P(t)}$. Here, *PMI*(k,t) score indicates how likely two songs k and t co-occur together, or how likely a target song k will be added into song t's playlist.

Given a playlist that has a set of l member songs, we measure *PMI* scores between the target song k and each of l member songs. Then, we apply $softmax$ to those *PMI* scores to obtain *PMI attentive scores*. Intuitively, the member song t that has a higher *PMI* score with candidate song k (i.e., co-occurs more with song k) will have a higher *PMI attentive score*. We draw scatter plots between *PMI attentive scores* and attentive scores generated by our proposed attention mechanism and its variations. Figure 4.13 shows the experimental results. We observe that the Pearson correlation ρ between the *PMI attentive scores* and the attentive scores generated by our attention mechanism is the highest (0.254). This result shows that our proposed attention tends to give higher scores to co-occurred songs, which is what we desire. The Pearson correlation results are also consistent with what was reported in Table 4.6.

Runtime comparison: To compare model runtimes, we used a Nvidia GeForce GTX 1080 Ti with a batch size of 256 and embedding size of 64. We do not report *MASR* and *AMASR*'s runtimes because their components are pretrained and fixed (i.e., there is no learning process/time). Figure 4.14 shows the runtimes (seconds per epoch) of our models and the baselines for each dataset. *MDR* only took 39 and 173 seconds per epoch in *30Music* and *AOTM*, respectively, while *MASS* took 88 and 375 seconds. *MDR*, one of the fastest models, was also competitive with *CML* and *MF-BPR*.

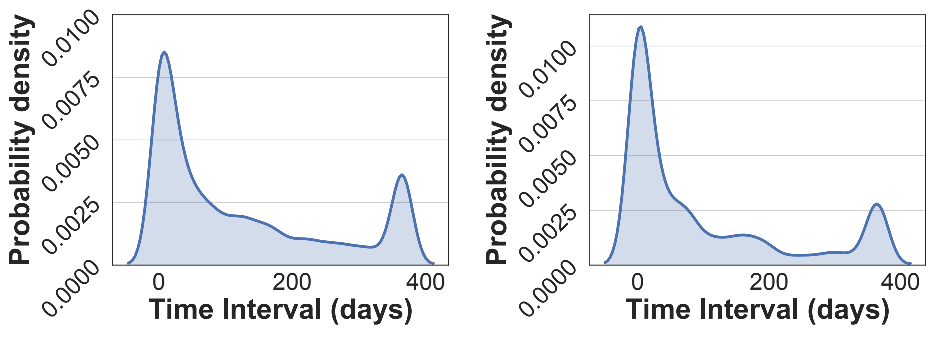
5

Modeling consumer behaviors with long-term and short-term dependencies

5.1 Introduction

Recommender Systems [114] have become the heart of many online applications such as e-commerce, music/video streaming services, social media, *etc*. Recommender systems proactively helped (i) users to explore new/unseen items, (ii) potentially the users stay longer on the applications, and (iii) companies increase their revenue.

Matrix Factorization techniques [23, 24, 74] extracted features of users and items to compute their similarity. Recently, deep neural network boosted performance of a recommender system by providing non-linearity which helped modeling complex relationships between users and items [26]. However, these prior works only focused on a user and a target item without considering the user's previously consumed items, some of which may be related to the target item. While some prior works [35, 36] largely premised on unordered user interactions, users' interests are intrinsically dynamic and evolving. Based on the observation, [40, 41, 42, 43, 47] followed two paradigms to capture a user's



(a) Video Games. (b) Toys and Games.

Figure 5.1: Density distribution of item-item similarity scores on Amazon *Video Games*, and *Toys and Games* datasets.

sequential pattern: (i) *short-term* item-item transitions, or (ii) *long-term* item-item transitions.

However, user's interests can be highly diverse, so modeling only either *short-term* or *long-term* user intent does not fully capture the user's preferences, producing less effective recommendation results. To illustrate the point, we conducted an empirical analysis on *Amazon Video Games*, and *Toys and Games* datasets. First, we represent each item by a multi-hot encoding, where item j is represented by a vector $t \in \mathbb{R}^m$, position $i = 1$ if user i consumed the current item, and m denotes the total number of users in a dataset. For each user, her consumed items are sorted in the chronological order. Then, we calculated a cosine similarity score between each item and each of its previously consumed items. Then we selected the largest cosine similarity score per item per user. Figure 5.1 presents the density distribution of the consumed time interval (*x-axis*) between each pair of item and its most similar previously consumed item. We observe that there exists a bimodal distribution, where one (*left*) peak lays at a relative short-term period and the other (*right*) peak locates in a long-term period. The observation confirms that both long-term and short-term preferences played important roles on the user's current purchasing intent. We observe the same phenomenon from the other four datasets described in Section 5.3.4.

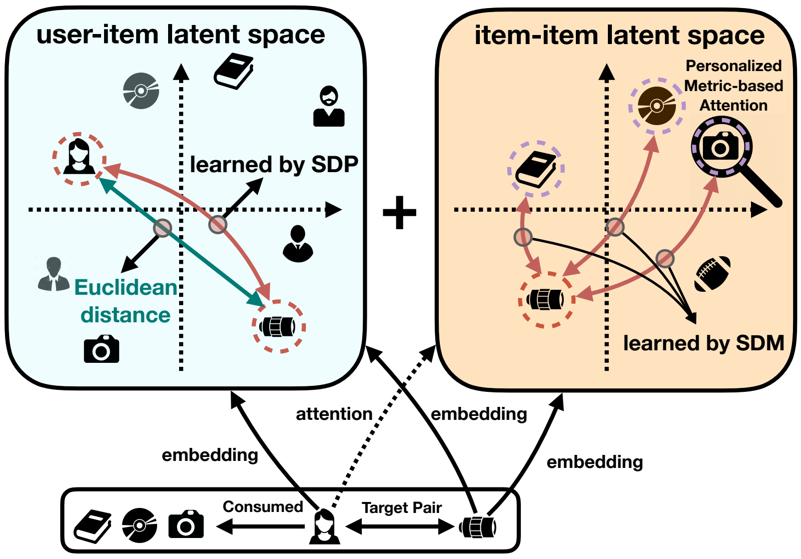


Figure 5.2: We consider a recommender as a signed distance approximator, and decompose the signed distance between a user and an item into two parts: the left box learns an explicitly signed distance between the user and item (i.e., the *camera lens*), the right box learns an implicitly signed distance between the user and the item via the user's recently consumed items (i.e., the *book*, *CD* and *camera*). Our novel personalized metric-based soft attention is applied to the consumed items to optimize their contributions to the output signed distance score. Then the two parts are combined to obtain a final score. Most of linear latent factor models are equivalent to simply measuring the linear Euclidean distance in the user-item latent space (shown as the green line).

In the following sections, we describe our two proposals for modeling consumer behaviors with long-term and short-term dependencies. Note that the two terms "user" and "consumer" are used interchangeably.

5.2 Recommending Products with a Neural Signed Distance Based Approach

In a perspective, we can view most of the recommendation models as a measurement of similarity or distance between a user and an item. For instance, the well known latent factor (i.e., matrix factorization) models [35] usually employ an inner product function to approximate the similarity between the user and the item. Although the latent factor models achieved competitive performance in some datasets, they did not correctly capture complex (i.e., non-linear) relationships between users and items because the inner product function follows limited linear nature.

Existing recommendation algorithms faced difficulties in finding good kernels for different data patterns [115], only focused on user-item latent space without considering the item-item latent space together [26, 31, 116, 117, 118], or required additional auxiliary information (e.g., item description, music content, reviews) [53, 54, 55, 56, 57]. To overcome the drawbacks, in this section, we aim to propose and build a deep learning framework to learn a non-linear relationship between a user and a target item by measuring a distance from the observed data. In particular, we propose *Signed Distance-based Deep Memory Recommender* (SDMR), which captures the long-term and short-term dependencies of the non-linear relationship of the user and item *explicitly* and *implicitly*, combines *explicitly* and *implicitly* measured relationship to produce a final distance score for the recommendation, and performs well in both general recommendation task and shopping basket-based recommendation task.

SDMR internally combines two signed distances, each of which is measured by our proposed *Signed Distance-based Perceptron* (SDP) and *Signed Distance-based Memory Network* (SDM). On one hand, SDP explicitly measures a global non-linear signed distance between the user and the item. Many existing models [23, 74] rely on a pre-defined metric such as Euclidean distance (the green line in Figure 5.2) which is much more limited than the customized non-linear signed distance learned from the data (the red curves in Figure 5.2). On the other hand, SDM implicitly measures a non-linear signed distance between the user and the item via the user's recently consumed items, and captures the short-term dependencies of the user's interest. SDM is similar to the item neighborhood-based recommender [90, 103] in nature. However, it is more advanced in several aspects, as shown in the right side of Figure 5.2. First, SDM only focuses on a set of recently consumed items of the target user (*e.g.*the *book*, *CD* and *camera* in Figure 5.2) as context items to encode the user's current taste. Second, it employs additional memories to learn a novel personalized metric-based attention on the consumed items. The goal of our proposed attention is to compute weights of each consumed item *w.r.t.* the target item (i.e., the *camera lens*). In the example, the attention module assigns higher weights on the *camera* and lower weights on the *book* and *CD*. Unlike our approach, most of the existing neighborhood-based models consider contribution of consumed items to the target item equally, leading to suboptimal results. Last but not the least, we update the attention weights via a gated multi-hop to build a long-term memory within SDM. This multi-hop design helps refine our attention module and produces more accurate attentive scores.

In the following sections, we describe the problem definition, and our proposed models. Then, we present the experimental settings and discuss about the experimental results.

5.2.1 Problem Definition

In this section, we describe two recommendation problems: (i) general recommendation task; and (ii) shopping basket-based recommendation task. In the following sections, we focus on solving them.

General recommendation task: Given a whole item set $V = \{v_1, v_2, ..., v_{|V|}\}$, and a whole user set $U = \{u_1, u_2, ..., u_{|U|}\}$. Each user $u_i \in U$ may consume several items $\{v_{i1}, v_{i2}, ..., v_{ik}\}$ in V, denoted as a set of context items c. In this task, given a user's previously consumed items, a recommendation model predicts a next target item v_j that user u_i may prefer, denoting this task as estimating $P(u_i, v_j|c)$. Note that some existing works assume independent relationships between v_j and context items in the set ***c***, leading to $P(u_i, v_j|c) = P(u_i, v_j)$ [26, 74]. In our work, we model the u_i's preference on v_j in two steps: (i) an explicit preference of u_i on v_j in a signed distance based perceptron, and (ii) an implicit preference of u_i on v_j via summing attentive effects of context items toward target item v_j in a signed distance based memory network.

Shopping Basket-based recommendation task: This problem is based on the fact that users go shopping offline/online and add some items into a basket/cart together. Each shopping basket/cart is seen as a transaction, and each user may shop once or multiple times, leading to one or multiple transactions. Let $T^{(u)} = \{t_1, t_2, ..., t_{|T^{(u)}|}\}$ as a set of the user u's transactions, where $|T^{(u)}|$ denotes the number of user u's transactions. Each transaction $t_i = \{v_1, v_2, ..., v_{|t_i|}\}$ consists of several items in the whole item set V. In this problem, it is assumed that all the items in t_i are inserted into the same basket at the same time, ignoring the actual order of the items being inserted and considering t_i's transaction time as each item's insertion time. Given a target item $v_j \in t_i$, the rest of the items in t_i will be seen as the context items of v_j, denoted as c (i.e. $c = t_i n\{v_j\}$). Then, given the set of context items c, a recommendation model predicts a conditional probability $P(u, v_j|c)$, which is interpreted as the conditional probability that u will add the item v_j into the same basket with the other items c.

Both of the recommendation tasks above are popular in the literature [26, 39, 40, 119]. The *general recommendation task* differs from the *shopping basket-based recommendation task* because there is no specific context items of the target item in the *general recommendation task*. Note that the two tasks are *personalized* recommendation problems. In fact, there are *non-personalized* recommendation problems such as *session-based recommendation* [43], where users (i.e. user IDs) are not available in transactions. However, in this chapter, we focus on *personalized* recommendation tasks because they are more preferred in the literature [39, 40, 119].

5.2.2 Method

Our proposed *Signed Distance-based Deep Memory Recommender* (SDMR) consists of two major components: *Signed Distance-based Perceptron* (SDP) and *Signed Distance-*

based Memory network (SDM). We first describe an overview of our models as follows:

- Given a target user i and a target item j as two one-hot vectors, we pass the two vectors through the user and item embedding spaces to get user embedding u_i and item embedding v_j.
- On one hand, our proposed *Signed Distance-based Perceptron* (SDP) will measure a signed distance score between u_i and v_j by a multi-layer perceptron network.
- On the other hand, given target user i, target item j, and the user i's recently consumed context items s as the input, our *Signed Distance-based Memory network* (SDM) will measure a signed distance score between user i and item j via attentive distances between context items s and target item j.
- Then, the *Signed Distance-based Deep Memory Recommender* (SDMR) model will measure a total distance between user i and item j by learning a combination of SDP and SDM. The smaller the total distance is, the more likely user i will consume item j.

Next, we describe SDP, SDM, and SDMR in detail.

Signed Distance-based Perceptron (SDP)

We first propose *Signed Distance-based Perceptron* (SDP) that *explicitly* learns a signed distance between a target user i and a target item j. An illustration of SDP is shown in Figure 5.3. Let the embedding of a target user i be $\boldsymbol{u}_i \in \mathbb{R}^d$, and the embedding of a target item j be $\boldsymbol{v}_j \in \mathbb{R}^d$, where d is the number of dimensions in each embedding. First, SDP takes a concatenation of these two embeddings as the input and proceeds as follows:

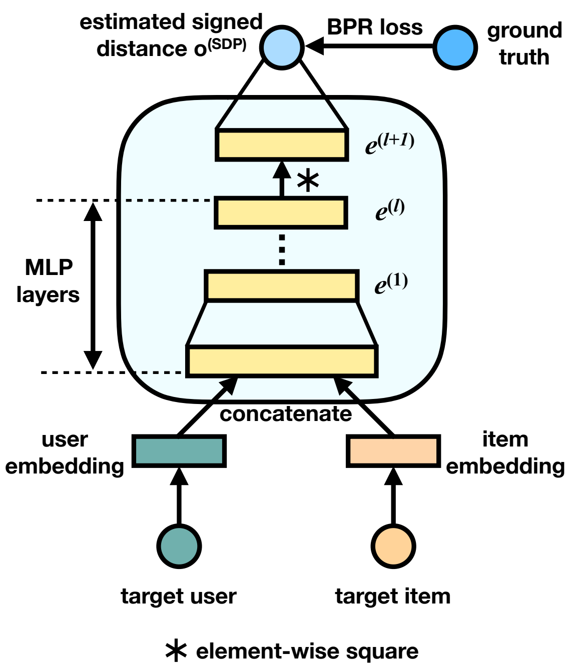


Figure 5.3: The illustration of our SDP model.

$$e^{(1)} = f_1(\mathbf{W}^{(1)} \begin{bmatrix} \boldsymbol{u}_i \\ \boldsymbol{v}_j \end{bmatrix} + \boldsymbol{b}^{(1)}) \tag{5.1}$$

$$e^{(2)} = f_2(\mathbf{W}^{(2)} e^{(1)} + \boldsymbol{b}^{(2)}) \tag{5.2}$$

$$\cdots \tag{5.3}$$

$$e^{(\ell)} = f_\ell(\mathbf{W}^{(\ell)} e^{(\ell-1)} + \boldsymbol{b}^{(\ell)}) \tag{5.4}$$

$$e^{(\ell+1)} = square(e^{(\boldsymbol{\ell})}) \tag{5.5}$$

$$o^{(SDP)} = \boldsymbol{w}^{(o)^\top} e^{(\ell+1)} + \boldsymbol{b}^{(o)} \tag{5.6}$$

where $f_l(\cdot)$ refers to a non-linear activation function at the layer l^{th} (e.g. `sigmoid`, `ReLu` or `tanh`), and $square(\cdot)$ denotes an element-wise square function (e.g $square([2, 3]) = [6, 9]$). Through experimental results, we choose `tanh` as the activation function because it yields slightly better results than `ReLu`. From now on, we will use $f(\cdot)$ to denote the `tanh` function. It can be easily observed that Eq. (5.1) – (5.4) form a trivial Multi-

Layer Perceptron (MLP) network, which is a popular design [26, 120] to learn a complex and non-linear interaction between user embedding $\boldsymbol{u}_i$ and item embedding $\boldsymbol{v}_j$. Our new design starts at Eq. (5.5) – Eq. (5.6). In Eq. (5.5), we apply the element-wise squared function $square(\cdot)$ to the output vector $\boldsymbol{e}^{(l)}$ of the MLP and obtain a new output vector $\boldsymbol{e}^{(l+1)}$. Next, in Eq. (5.6), we use a fully connected layer $\boldsymbol{w}^{(o)}$ to combine different dimensions in $\boldsymbol{e}^{(l+1)}$ and yields a final distance value $o^{(SDP)}$. Our idea of using $\boldsymbol{w}^{(o)}$ in here is that after applying the element-wise square function $square(\cdot)$ in Eq. (5.5), all the dimensions in $\boldsymbol{e}^{(l+1)}$ will be non-negative. Thus, we consider each dimension of $\boldsymbol{e}^{(l+1)}$ as a distance value. The edge weights $\boldsymbol{w}^{(o)}$ will then be used to combine those distant dimensions to provide a more fine-grained distance.

We note that SDP can be reduced to a squared Euclidean distance with the following setting: at Eq. (5.1), $\mathbf{W}^{(1)} = [\mathbb{I}, -\mathbb{I}]$ with $\mathbb{I}$ denotes an identity matrix and so $\mathbf{W}^{(1)} \begin{bmatrix} \boldsymbol{u}_i \\ \boldsymbol{v}_j \end{bmatrix} = \boldsymbol{u}_i - \boldsymbol{v}_j$; the activation $f(\cdot)$ is an identity function; the number of MLP layers $\ell = 1$; the edge-weights layer at Eq. (5.6): $\boldsymbol{w}^{(o)} = \mathbf{1}$ (e.g. the all-ones matrix), bias $\boldsymbol{b}^{(o)} = \mathbf{0}$. Note that if $\boldsymbol{w}^{(o)}$ in Eq. (5.6) is an all-negative layer, it will yield a negative value, which we name as a signed distance[1] score. If we see each user $\boldsymbol{i}$ as a point in multi dimensional space, and the user's preference space is defined by a boundary Ω, we can interpret this signed distance score as follows: When the item $\boldsymbol{j}$ is out of the user $\boldsymbol{i}$'s preference boundary Ω, the distance $d(\boldsymbol{i}, \boldsymbol{j})$ between them is positive (i.e. $d(\boldsymbol{i}, \boldsymbol{j})$ ¿ 0) and it reflects that user $\boldsymbol{i}$ does not prefer item $\boldsymbol{j}$. When the distance between user $\boldsymbol{i}$ and item $\boldsymbol{j}$ is shortened and $\boldsymbol{j}$ is right on the boundary Ω, the distance between them is zero and it indicates user $\boldsymbol{i}$ likes item $\boldsymbol{j}$. As $\boldsymbol{j}$ is coming inside Ω, the distance between them becomes negative and reflects a higher preference of user $\boldsymbol{i}$ on item $\boldsymbol{j}$. In short, we can see SDP as a signed distance function, which could learn a complex signed distance between a

[1]https://en.wikipedia.org/wiki/Signed_distance_function

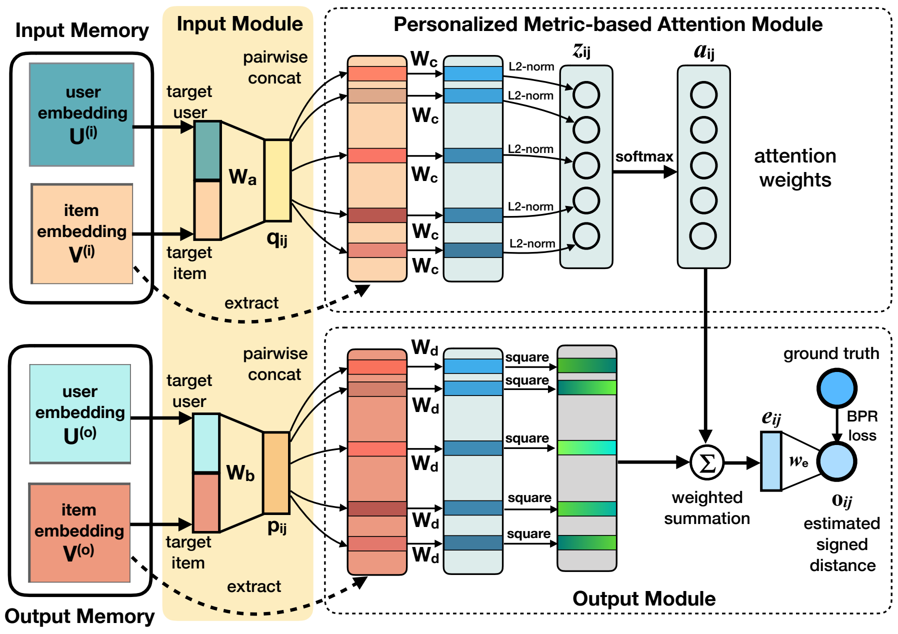


Figure 5.4: The illustration of single-hop SDM, which consists of a memory module, an input module, an attention module, and an output module.

user and an item via a MLP architecture with non-linear activations and an element-wise square function $square(\cdot)$. In the recommendation domain, the signed distances will provide more fine-grained distance values, thus, reflecting a user' preferences on items more accurately (i.e. accurately rank items for the user).

Signed Distance-based Memory Network (SDM) We propose a multi-hop memory network, *Signed Distance-based Memory network* (SDM), to model *implicit* preference of a user on the target item via the user's previously consumed items (i.e., context items). The implicit preference is represented as a signed distance. First, we describe a single-hop SDM, and then describe how to extend it into a multi-hop design. Following the traditional architecture of a memory network [100, 104, 121], our proposed single-hop SDM has four main components: a memory module, an input module, an attention module, and an output module. The overview of SDM's architecture is presented in Figure 5.4. We will go into details of each SDM's module as follows:

Memory Module: We maintain two memories called input memory and output memory. The input memory contains two embedding matrices $\mathbf{U}^{(i)} \in \mathbb{R}^{M \times d}$ and $\mathbf{V}^{(i)} \in \mathbb{R}^{N \times d}$, where M and N are the number of users and the number of items in the system, respectively. d denotes the embedding size of each user and each item. Similarly, the output memory also contains two embedding matrices $\mathbf{U}^{(o)} \in \mathbb{R}^{M \times d}$ and $\mathbf{V}^{(o)} \in \mathbb{R}^{N \times d}$. As shown in Figure 5.4, the input memory will be used to calculate attention weights of a user's consumed items (i.e., context items), whereas the output memory will be used to measure a final signed distance between the target user and the target item via the user's context items.

Given a target user i, a target item j and a set of user i's consumed items as context items $\mathcal{T}_j^i$, the output of this module is the embeddings of user i, item j, and all context items $k \in \mathcal{T}_j^i$: $(\boldsymbol{u}_i, \boldsymbol{v}_j$, ¡$\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k$¿). Since this module has a separated input memory and output memory, we obtain $(\boldsymbol{u}_i^{(i)}, \boldsymbol{v}_j^{(i)}$, ¡$\boldsymbol{v}_1^{(i)}, \boldsymbol{v}_2^{(i)}, ..., \boldsymbol{v}_k^{(i)}$¿) as the output of the input memory, and $(\boldsymbol{u}_i^{(o)}, \boldsymbol{v}_j^{(o)}$, ¡$\boldsymbol{v}_1^{(o)}, \boldsymbol{v}_2^{(o)}, ..., \boldsymbol{v}_k^{(o)}$¿) as the output of the output memory. It is obvious that $\boldsymbol{u}_i^{(i)}$ is the i-th row of $\mathbf{U}^{(i)}$, $\boldsymbol{v}_j^{(i)}$ and $\boldsymbol{v}_k^{(i)}$ are the corresponding j-th and k-th row of $\mathbf{V}^{(i)}$. A similar explanation is applied to $\boldsymbol{u}_i^{(o)}$ $\boldsymbol{v}_j^{(o)}$, and $\boldsymbol{v}_k^{(o)}$.

Input Module: The goal of the input module is to form a non-linear combination between the target user embedding and the target item embedding. Given the target user embedding $\boldsymbol{u}_i^{(i)}$ and the target item embedding $\boldsymbol{v}_j^{(i)}$ from the input memory in the memory module, following the widely adopted design in multimodal deep learning work [99, 122], the input module simply concatenates the two embeddings, and then applies a fully connected layer with a non-linear activation $f(\cdot)$ (i.e. `tanh` function) to obtain a coherent hidden feature vector as follows:

$$\boldsymbol{q}_{ij} = f\Big(\mathbf{W}_a \begin{bmatrix} \boldsymbol{u}_i^{(i)} \\ \boldsymbol{v}_j^{(i)} \end{bmatrix} + \boldsymbol{b}_a\Big) \tag{5.7}$$

where $\mathbf{W}_a \in \mathbb{R}^{d \times 2d}$ is the weights of input module. Note that $q_{ij} \in \mathbb{R}^d$ can be seen as a query embedding in Memory Network [104].

Similarly, if the inputs of the input module are the target user embeddings $\boldsymbol{u}_i^{(o)}$ and the target item embeddings $\boldsymbol{v}_j^{(o)}$ from the output memory, we can form a non-linear combination between $\boldsymbol{u}_i^{(o)}$ and $\boldsymbol{v}_j^{(o)}$ (i.e. an output query), denoted as $\boldsymbol{p}_{ij}$, as follows:

$$\boldsymbol{p}_{ij} = f\Big(\mathbf{W}_b \begin{bmatrix} \boldsymbol{u}_i^{(o)} \\ \boldsymbol{v}_j^{(o)} \end{bmatrix} + \boldsymbol{b}_b\Big) \tag{5.8}$$

Attention Module: The goal of the attention module is to assign attentive scores to different context items (or candidates) given the combined vector (or a query) $\boldsymbol{q}_{ij}$ of the target user i and target item j obtained in Eq. (5.7). First, we calculate the squared $\mathcal{L}2$ distance between $\boldsymbol{q}_{ij}$ and each candidate item $\boldsymbol{v}_k^{(i)}$ as follows:

$$z_{ijk} = \Big\|f\Big(\mathbf{W}_c \begin{bmatrix} \boldsymbol{q}_{ij} \\ \boldsymbol{v}_k^{(i)} \end{bmatrix} + \boldsymbol{b}_c\Big)\Big\|_2^2 \tag{5.9}$$

where $||\cdot||_2$ refers to the $\mathcal{L}2$ distance (or Euclidean distance), which is widely used in previous works to measure similarity among items [39] or between users and items [92]. To better understand our intuition in Eq. (5.9), we will break it into smaller parts and explain them. First, similar to the intuition of Eq. (5.7), we have $f\Big(\mathbf{W}_c \begin{bmatrix} \boldsymbol{q}_{ij} \\ \boldsymbol{v}_k^{(i)} \end{bmatrix} + \boldsymbol{b}_c\Big)$ component to define a non-linear combination between the input query $\boldsymbol{q}_{ij}$ and each context item embeddings $\boldsymbol{v}_{\boldsymbol{k}}^{(i)}$. Then, $||\cdot||_2^2$ will measure the squared $\mathcal{L}2$ distance of the combined vector. It is worth to note that with a following setting: $\boldsymbol{W}_a = [\mathbf{0}, \mathbb{I}]$ where $\mathbb{I}$ refers to an identity matrix and $\mathbf{0}$ is an all-zeros matrix; $f(\cdot)$ is an identity function; $\boldsymbol{W}_c = [\mathbb{I}, -\mathbb{I}]$; bias terms $\boldsymbol{b}_a = \boldsymbol{b}_c = 0$. Then, in Eq. (5.7), $\boldsymbol{q}_{ij} = f\Big(\mathbf{W}_a \begin{bmatrix} \boldsymbol{u}_i^{(i)} \\ \boldsymbol{v}_j^{(i)} \end{bmatrix} + \boldsymbol{b}_a\Big) = \boldsymbol{v}_j^{(i)}$;

in Eq. (5.9), $f\Big(\mathbf{W}_c \begin{bmatrix} \boldsymbol{q}_{ij} \\ \boldsymbol{v}_k^{(i)} \end{bmatrix} + \boldsymbol{b}_c\Big) = \boldsymbol{v}_j^{(i)} - \boldsymbol{v}_k^{(i)}$, and $z_{ijk} = ||(\boldsymbol{v}_j^{(i)} - \boldsymbol{v}_k^{(i)})||_2^2$, which simply generalizes a squared $\mathcal{L}2$ distance between the target item j and the context item k. Additionally, with another setting: $\boldsymbol{W}_a$ = $[\nVdash, -\nVdash]$; $f(\cdot)$ is an identity function; $\boldsymbol{W}_c$ = $[\nVdash, \nVdash]$; bias terms $\boldsymbol{b}_a = \boldsymbol{b}_c = 0$. Then, in Eq. (5.7), $\boldsymbol{q}_{ij} = f\Big(\mathbf{W}_a \begin{bmatrix} \boldsymbol{u}_i^{(i)} \\ \boldsymbol{v}_j^{(i)} \end{bmatrix} + \boldsymbol{b}_a\Big) = \boldsymbol{u}_i^{(i)} - \boldsymbol{v}_j^{(i)}$, in Eq. (5.9), $f\Big(\mathbf{W}_c \begin{bmatrix} \boldsymbol{q}_{ij} \\ \boldsymbol{v}_k^{(i)} \end{bmatrix} + \boldsymbol{b}_c\Big) = \boldsymbol{u}_i^{(i)} - \boldsymbol{v}_j^{(i)} + \boldsymbol{v}_k^{(i)}$, and $z_{ijk} = ||(\boldsymbol{v}_k^{(i)} + \boldsymbol{u}_i^{(i)} - \boldsymbol{v}_j^{(i)})||_2^2$, which simply generalizes a squared $\mathcal{L}2$ distance between the target item j and the context item k where the user i plays as a translator [41]. The two examples above show that our proposed design can learn a more generalized distance between target and context items.

The output squared $\mathcal{L}2$ distance in Eq. (5.9) will show how similar the target item j and the context item k are. The lower the distance score is, the more similar two items j and k are. Next, we use the Softmax function to normalize and obtain attentive score between j and k as follows:

$$a_{ijk} = \frac{exp(-z_{ijk})}{\sum_{p \in \mathcal{T}_j^i} exp(-z_{ijp})} \tag{5.10}$$

where $\mathcal{T}_j^i$ is the set of user $\boldsymbol{i}$'s neighborhood items. The **minus** sign in Eq. (5.10) is used to assign a higher attention score for a lower distance between two items (j, k).

We note that the $\mathcal{L}2$ distance (or Euclidean distance) satisfies four conditions of a metric [1]. While the crucial triangle inequality property of a metric was shown to provide a better performance compared to the inner product [92, 93, 123] in recommendation domains, to our best of knowledge, most of existing attention designs [112, 113, 124, 125, 126, 127, 128] adopted the inner product for measuring attentive scores. Hence, this proposed attention design is the **first attempt** to bring metric properties into the attention

[1]https://en.wikipedia.org/wiki/Metric_(mathematics)

mechanism.

Similar to [129], we limit the number of considering context items by choosing the user $\boldsymbol{i}$'s $\boldsymbol{s}$ most recently consumed items before target item $\boldsymbol{j}$ as the context items of target item $\boldsymbol{j}$. Here, $\boldsymbol{s}$ can be selected via tuning with a development dataset. The soft attention vector containing attentive contribution scores of $\boldsymbol{s}$ context items toward the target item $\boldsymbol{j}$ of a user $\boldsymbol{i}$ is given as follows:

$$\boldsymbol{a}_{ij} = \begin{bmatrix} a_{ij1} \\ \dots \\ a_{ijs} \end{bmatrix} \tag{5.11}$$

Output Module: Given the attentive scores $\boldsymbol{a}_{ij}$ in Eq.(5.11) and the combined vector $\boldsymbol{p}_{ij} \in \mathbb{R}^d$ of the user embedding $\boldsymbol{u}_i^{(o)}$ and item embedding $\boldsymbol{v}_j^{(o)}$ from the output memory $\boldsymbol{U}^{(o)}$ and $\boldsymbol{V}^{(o)}$, the goal of this output module is to measure a total output distance $\boldsymbol{o}_{ij}^{(SDM)}$ between the output target item embeddings $\boldsymbol{v}_j^{(o)}$ and all the user $\boldsymbol{i}$'s output context item embeddings $\boldsymbol{v}_k^{(o)}(k \in T_j^i)$ using attention weights $\boldsymbol{a}_{ij}$ and the output query $\boldsymbol{p}_{ij}$ as follows:

$$o_{ij}^{(SDM)} = \boldsymbol{w}_e^\top \boldsymbol{e}_{ij} + b_e \tag{5.12}$$

where $\boldsymbol{e}_{ij} \in \mathbb{R}^d$ is calculated as follows:

$$\boldsymbol{e}_{ij} = \sum_{k \in \mathcal{T}_j^i} \boldsymbol{a}_{ijk} \times square\Big(f\Big(\mathbf{W}_d \begin{bmatrix} \boldsymbol{p}_{ij} \\ \boldsymbol{v}_k^{(o)} \end{bmatrix} + \boldsymbol{b}_d\Big)\Big) \tag{5.13}$$

In here, let $\boldsymbol{r}_{ijk} = f\Big(\mathbf{W}_d \begin{bmatrix} \boldsymbol{p}_{ij} \\ \boldsymbol{v}_k^{(o)} \end{bmatrix} + \boldsymbol{b}_d\Big)$. Similar to the previously discussed intuition in Eq (5.9), $\boldsymbol{r}_{ijk}$ is a flexible combination between $\boldsymbol{p}_{ij}$ and each output context item embeddings $\boldsymbol{v}_k^{(o)}$; $square(\cdot)$ is an element-wise squared function. Our idea in Eq. (5.12), (5.13)

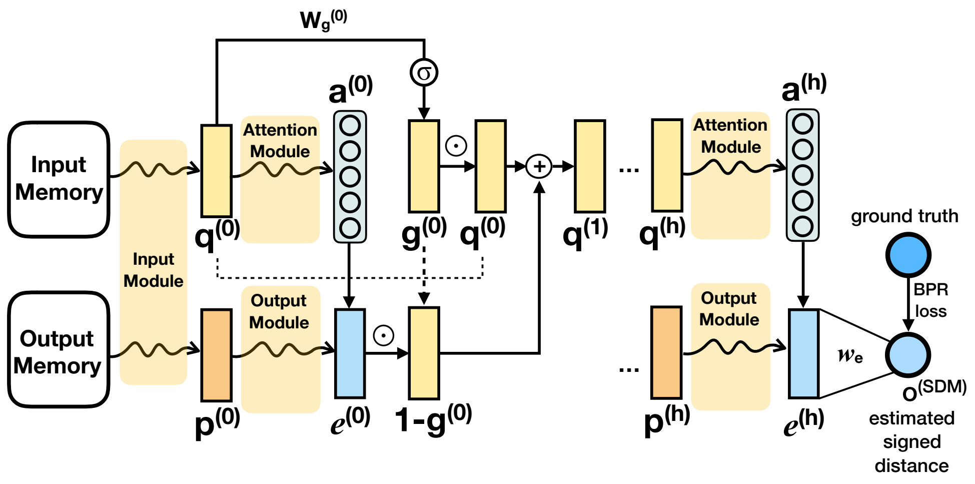


Figure 5.5: The illustration of our multi-hop SDM.

is similar to the idea in Eq. (5.5), (5.6) of the SDP model. First, in Eq. (5.13), each context item k will attentively contribute to the target item j via a squared Euclidean measure. Second, in Eq. (5.12), each non-negative dimension in e_{ij} will be considered as a distance dimension and we use an edge-weights layer $\boldsymbol{w}_e$ to combine them flexibly. When there is only one context item in $\mathcal{T}_j^i$, then in Eq. (5.13), the attention score $\boldsymbol{a}_{ijk}$=1.0, leading to $\boldsymbol{e}_{ij} = square(\boldsymbol{r}_{ijk})$, which is similar to Eq. (5.5). In this case, SDM will measure the distance between target item j and context item k in the same way as SDP model does. Note that Eq. (5.13) is similar to Eq. (5.6) so SDM can also learn a signed distance value, which also provides a more fine-grained distance compared to a general distance value.

Multi-hop SDM: Inspired by previous work [104] where the multi-hop design helped to refine the attention module in Memory Network, we also integrate multiple hops to further extend our SDM model to build a deeper network (Figure 5.5). As the gated multi-hop design [121] was shown to perform better than the original multi-hop design with a simple residual connection in [104], we employ this gated memory update from hop to hop as follows:

$$\boldsymbol{g}^{(h-1)} = \sigma(\mathbf{W}_g^{(h-1)}\boldsymbol{q}^{(h-1)} + \boldsymbol{b}_g^{(h-1)}) \tag{5.14}$$

$$\boldsymbol{q}^{(h)} = (1 - \boldsymbol{g}^{(h-1)}) \odot \boldsymbol{e}^{(h-1)} + \boldsymbol{g}^{(h-1)} \odot \boldsymbol{q}^{(h-1)} \tag{5.15}$$

where $\boldsymbol{q}^{(h-1)}$ is the input query embedding as shown in Eq. (5.7) at hop $h-1$, $\mathbf{W}_g^{(h-1)}$ and bias $\boldsymbol{b}_g^{(h-1)}$ are hop-specific parameters, σ is the sigmoid function, $e^{(h-1)}$ is the output of Eq. (5.13) at hop $h-1$, $\boldsymbol{q}^{(h)}$ is the input query embedding at the next hop h. So the attention could be updated at hop h accordingly using $\boldsymbol{q}^{(t)}$ as follows:

$$\alpha_{ijk}^{(h)} = \frac{exp(-z_{ijk}^{(h)})}{\sum_{p\in\mathcal{T}_j^i} exp(-z_{ijp}^{(h)})} \tag{5.16}$$

where $z_{ijk}^{(h)}$ is measured by:

$$z_{ijk}^{(h)} = \left\| f\left(\mathbf{W}_c^{(h)} \begin{bmatrix} \boldsymbol{q}_{ij}^{(h)} \\ \boldsymbol{v}_k^{(i)} \end{bmatrix} + \boldsymbol{b}_c\right)\right\|_2^2 \tag{5.17}$$

The multi-hop architecture with gated design further refines the attention for different users based on the previous output from hop to hop. Hence, if the final hop is h then the SDM model with h hops, denoted as *SDM-h*, will use $\boldsymbol{a}_{ij}^{(h)}$ to yield a final signed distance score as follows:

$$o_{ij}^{(SDM-h)} = \boldsymbol{w}_e^\top \boldsymbol{e}_{ij}^{(h)} + b_e^{(h)} \tag{5.18}$$

where $\boldsymbol{e}_{ij}$ is calculated as:

$$\boldsymbol{e}_{ij}^{(h)} = \sum_{k\in\mathcal{T}_j^i} \boldsymbol{a}_{ijk}^{(h)} \times square\Big(f\Big(\mathbf{W}_d^{(h)} \begin{bmatrix} \boldsymbol{p}_{ij}^{(h)} \\ \boldsymbol{v}_k^{(o)} \end{bmatrix} + \boldsymbol{b}_d^{(h)}\Big)\Big) \tag{5.19}$$

Weight constraints in multi-hop SDM model: To save memory, we use the global weight constraint in multi-hop SDM. Particularly, input memory $\boldsymbol{U}^{(i)}, \boldsymbol{V}^{(i)}$ and output memory $\boldsymbol{U}^{(o)}, \boldsymbol{V}^{(o)}$ are shared among different hops. All the weights are shared from hop to hop $\boldsymbol{W}_a^{(1)} = \boldsymbol{W}_a^{(2)} = ... = \boldsymbol{W}_a^{(h)}$; $\boldsymbol{W}_b^{(1)} = \boldsymbol{W}_b^{(2)} = ... = \boldsymbol{W}_b^{(h)}$; $\boldsymbol{W}_c^{(1)} = \boldsymbol{W}_c^{(2)} = ... = \boldsymbol{W}_c^{(h)}$; $\boldsymbol{W}_d^{(1)} = \boldsymbol{W}_d^{(2)} = ... = \boldsymbol{W}_d^{(h)}$; and so do all bias terms. The gate weights are also global weights: $\boldsymbol{W}_g^{(1)} = \boldsymbol{W}_g^{(2)} = ... = \boldsymbol{W}_g^{(h)}$.

Signed Distance-based Deep Memory Recommender (SDMR) Now we propose Signed Distance-based Deep Memory Recommender (SDMR), a hybrid network that combines SDP and SDM. The first approach to combine them is to employ a weighted summation of the output scores from SDP and SDM as follows:

$$o = \beta o^{(\text{SDP})} + (1 - \beta) o^{(\text{SDM})} \tag{5.20}$$

where $o^{(\text{SDP})}$ is the signed distance score obtained at Eq. (5.6), $o^{(\text{SDM})}$ is the signed distance score obtained at Eq. (5.18), and $\beta \in [0, 1]$ is a hyper-parameter to control the contribution of SDP and SDM. When β=0, SDMR becomes SDM. When β=1, SDMR becomes SDP.

However, to avoid tuning an additional hyper-parameter β, we do not use Eq. (5.20) for SDMR. Instead, we let SDMR self-learns the combination of SDM and SDM as follows:

$$o = ReLU\left(\boldsymbol{w}_u^\top \begin{bmatrix} \boldsymbol{e}^{(\ell+1)} \\ \boldsymbol{e}^{(h)} \end{bmatrix} + b_u\right) \tag{5.21}$$

where $\boldsymbol{e}^{(\ell+1)}$ is the final layer embedding from SDP and is obtained at Eq. (5.5), $\boldsymbol{e}^{(h)}$ is the final hop output from the multi-hop SDM obtained at Eq. (5.19). We note that SDP and SDM are first pre-trained separately using the BPR loss function (see the next section). Then, we obtain $\boldsymbol{e}^{(\ell+1)}$ from SDP, and $\boldsymbol{e}^{(h)}$ from SDM, and keep them fixed in

Eq. (5.21) to learn $\boldsymbol{w}_u$ and b_u. We use `ReLU` in Eq. (5.21) because `ReLU` encourages sparse activations and helps to reduce over-fitting when combining the two components SDP and SDM.

Loss Functions We adopt the Bayesian Personalized Ranking (BPR) as our loss function, which is similar to the idea of AUC (area under the curve):

$$\mathcal{L} = \underset{\theta}{\operatorname{argmin}} \Big(- \sum_{(u,i^+,i^-)} \log \sigma(o_{ui^-} - o_{ui^+}) + \lambda \|\theta\|^2 \Big) \tag{5.22}$$

where we uniformly sample tuples in a form of (u, i^+, i^-) for user u with positive item (consumed) i^+ and negative item (unconsumed) i^-. λ is a hyper-parameter to control the regularization term, and $\sigma(\cdot)$ is the sigmoid function. Note that other pairwise probability functions could be plugged in Eq. (5.22) to replace $\sigma(\cdot)$. Both SDP and SDM are end-to-end differentiable since we uses soft attention over the output memory. Hence, we can utilize back-propagation to learn our models with stochastic gradient descent or Adam [130].

5.2.3 Experimental Settings

We evaluate our SDP, SDM, SDMR models against ten state-of-the-art baselines in two recommendation tasks: (i) *general recommendation task*, and (ii) *shopping basket-based recommendation task*. We mainly aim to answer the following research questions (RQs):

- **RQ1:** How do SDP, SDM, and SDMR perform compared to other state-of-the-art models in both general recommendation task and shopping basket-based recommendation task?
- **RQ2:** Why/How does the multi-hop design help to improve the proposed models' performance?

Table 5.1: Statistics of the four datasets in the general recommendation task.

Statistics	ML-100k	ML-1M	Netflix	Epinions
# of users	943	6,040	1,888	23,137
# of items	1,682	3,706	3,724	23,585
# of interactions	100,000	1,000,209	103,254	461,982
Density (%)	6.3%	4.5%	1.5%	0.08%

Datasets: We compare our models against the baselines in different recommendation tasks as follows:

General recommendation task: In this task, we evaluate our proposed models and state-of-the-art methods using different datasets with various density levels as follows:

- **Movielens** [68]: It is a widely adopted benchmark dataset for collaborative filtering evaluation. We use two versions of this benchmark dataset, namely MovieLens100k (or ML-100k) and MovieLens1M (or ML-1M).
- **Netflix Prize** [1]: It is a real-world dataset collected by Netflix. This dataset was collected from 1999 to 2005, and consists of 463,435 users and 17,769 items with 56.9M of interactions. Since the dataset is extremely large, we subsample the Netflix dataset by randomly picking one-month data for evaluation.
- **Epinions** [131] [2]: It is an online rating dataset where users can share product feedback by giving explicit ratings and reviews.

In preprocessing preparation, we adopted a popular k-core preprocessing step [31, 88, 110] (with *k-core* = 5) to filter out inactive users with less than five ratings and items which are consumed by less than five users. Since ML-100k and ML-1M are already pre-processed, we only apply 5-core preprocessing step on the Netflix and Epinions datasets. We also binarize the rating scores as implicit feedback by converting all observed rating

[1]https://www.netflixprize.com/

[2]http://www.trustlet.org/downloaded_epinions.html

Table 5.2: Statistics of the two real-world transactional datasets in the shopping basket-based recommendation task.

Statistics	IJCAI-15	Tafeng
# of users	2,433	22,851
# of items	4,534	22,291
avg # of items in a transaction	6.28	9.28
# of generated instances	15,422	523,653
Density (%)	0.14%	0.10%

scores as positive interactions and the remaining as negative interactions. The statistics of the four datasets are summarized in Table 5.1.

Shopping basket-based recommendation task: We use two real-world transaction datasets as follows:

- **IJCAI-15** [1]: It consists of shopping logs of users from Tmall [2]. Since the original dataset is extremely large scale. We subsample IJCAI-15 by randomly picking 20k transactions for evaluation.
- **Tafeng** [3]: It is a grocery store transaction data. It contains four month transaction data from November 2000 to February 2001 by T-Feng supermarket.

In both IJCAI-15 and Tafeng datasets, each user behavior is logged under four types of actions: *click*, *add-to-cart*, *purchase*, and *add-to-favourite*. We consider all the four types as the *click* action. We only keep transactions with at least five items. This is because we will take one item out for testing, another item for development. In the remaining three items, one will be taken out as a target item and the two items will be used as the context items. Attentive scores will be assigned to the context items. In each of original transactions, we generate data instances of the format $< \mathbf{c}, v_c >$ where v_c is the target/predicting item and $\mathbf{c}$ is a set of all other items in the same transaction with

[1]https://tianchi.aliyun.com/datalab/dataSet.htm?id=1
[2]https://www.tmall.com
[3]http://stackoverflow.com/questions/25014904/download-link-for-ta-feng-grocery-dataset

v_c. In particular, in each transaction t, each time we pick one item out as a target item and leave the rest of items in t as corresponding context items. Subsequently, for each transaction t containing $|t|$ items, we can generate $|t|$ data instances. The statistics of the two transactional datasets are summarized in Table 5.2.

For an easy reference, we call (ML-100k, ML-1M, Netflix, Epinions) as *Group-1 dataset* and (IJCAI-15, Ta-Feng) as *Group-2 datasets*.

Baselines and State-of-the-art Methods We compared our proposed models against several strong baselines in the general recommendation task as follows:

- **Item**KNN [90]: It is an item neighborhood-based collaborative filtering method. It exploited cosine item-item similarities to produce recommendation results.
- **Bayesian Personalized Ranking (MF-BPR)** [111]: It is a state-of-the-art pairwise matrix factorization method for implicit feedback datasets. It minimizes the following loss function:
$$\sum_{i}\sum_{j^+,j^-} -log\sigma(u_i^T v_{j^+} - u_i^T v_{j^-}) + \lambda(||u_i||^2 + ||v_{j^+}||^2)$$
where (u_i, v_{j^+}) is a positive interaction and (u_i, v_{j^-}) is a negative sample.
- **Sparse LInear Method (**SLIM) [103]: It learns a sparse item-item similarity matrix by minimizing the squared loss $||A - AW||^2 + \lambda_1||W|| + \lambda_2||W||^2$, where A is a $m \times n$ user-item interaction matrix and W is a $n \times n$ sparse matrix of aggregation coefficients of context items.
- **Collaborative Metric Learning (CML)** [92]: It is a state-of-the-art collaborative metric-based model that utilizes Euclidean distance to measure similarities between users and items. For fair comparison, we learn CML with BPR loss by minimizing $-\sum_{i,j^+,j^-} log(\sigma(||u_i - v_{j^-}||_2^2 - ||u_i - v_{j^+}||_2^2))$, where $|| \cdot ||_2^2$ is a squared Euclidean distance, (u_i, v_{j^+}) is a positive interaction and (u_i, v_{j^-}) is a negative sample.
- **Neural Collaborative Filtering (NeuMF++)** [26]: It is a state-of-the-art matrix fac-

torization method using deep learning architecture. We use a pre-trained NeuMF to achieve its best performance, and denote it as NeuMF++.

- **Collaborative Memory Network (CMN++)** [29]: It is a state-of-the-art memory network based recommender. Its architecture follows traditional user neighborhood based collaborative filtering approaches. It adopts a memory network to assign attentive weights for other similar users.

Even though our proposed methods do not model the order of consumed items in the user's purchase history (e.g. rigid orders of items), since we consider latest s items as the context items to predict the next item, we still compare our models with some key sequential models to further show our models' effectiveness as follows:

- **Personalized Ranking Metric Embedding (PRME)** [39]:
  Given a user u, a target item j, and a previous consumed item k, it models a personalized first-order Markov behavior with two components: $d_{ujk} = \alpha||v_u - v_j||^2 + (1-\alpha)||v_k - v_j||^2$, where $||\cdot||_2^2$ is a squared $\mathcal{L}2$ distance. Then PRME is learned by minimizing BPR loss.

- **PRME_s:** It is our extension of PRME, where the distance between the target item j and the previous consumed item k is replaced by the average distance between j and each of previous s items: $d_{ujs} = \alpha||v_u - v_j||^2 + (1-\alpha)\frac{1}{|s|}\sum_{k \in s}||v_k - v_j||^2$. We use BPR loss to learn PRME_s.

- **Translation-based Recommendation (TransRec)** [41]: It uses first-order Markov and considers a user u as a translator of his/her previous consumed item k to a next item j. In another word, $prob(j|u,k) \propto \beta_j - d(u + v_k - v_j)$ where β_j is an item bias term, d is a distance function (e.g. $\mathcal{L}1$ or $\mathcal{L}2$ distance). We use $\mathcal{L}2$ distance because it was shown to perform better than $\mathcal{L}1$ [41]. TransRec is then learned with BPR loss.

- **Convolutional Sequence Embedding Recommendation**

(Caser) [42]: It is a state-of-the-art sequential model. It uses convolution neural network with many horizontal and vertical kernels to capture the complex relationships among items.

The strong sequential baselines above surpassed many other sequential models such as: TransRec outperformed FMC[40], FPMC [40], HRM [132]; Caser surpassed GRU4Rec [43] and Fossil [133], so we exclude them in our evaluation.

Comparison: In the general recommendation task, we compare our proposed models with all **ten** strong baselines listed above. In the shopping basket-based recommendation task, since the sequential models often work better than general recommendation-based models (see Table 5.3), we only compared our proposed models with sequential baselines. We name general recommendation baselines (i.e. ItemKNN, BPR, SLIM, CML, NeuMF++, CMN++) as *Group-1 baselines*, and call sequential baselines (i.e. PRME, PRME_s, TransRec, Caser) as *Group-2 baselines* for an easy reference.

Experimental Protocol: We adopt the widely used *leave-one-out* setting [26, 120], in which for each user, we reserve her last interaction as the test sample. If there are no timestamps available in the dataset, then the test sample is randomly drawn. Among the remaining data, we randomly hold one interaction for each user to form the development set, while all others are utilized as the training set. Since it is very time-consuming and unnecessary to rank all the unobserved items for each user, we follow the standard strategy to randomly sample 100 unobserved items for each user. Then, we rank them together with the test item [26, 35].

Assigning item orders: Sequential models need rigid orders of consumed items but consumed items in the same transaction (in IJCAI-15 and TaFeng datasets) are assigned the same timestamp of the transaction containing these items. Hence, we assigned the item timestamps where the orders of items are kept as in the original dataset. This may give credits to sequential models but not our methods (because our methods will use all consumed items in the same transaction as context items and do not model the item orders).

Hyper-parameters selection: We perform a grid search for the embedding size from $\{8, 16, 32, 64, 128\}$ and regularization terms from $\{0.1, 0.01, 0.001, 0.0001, 0.00001\}$ in all the models. We select the best number of hops for CMN++ and our SDM from $\{1, 2, 3, 4\}$. In NeuMF++, we select the best number of MLP layers from $\{1, 2, 3\}$. In our models, we fix the batch size to 256. We adopt Adam optimizer [130] with a fixed learning rate of 0.001. Similar to CMN++ and NeuMF++, the number of negative samples is set to 4. We use one layer perceptron for SDP (more complex datasets may need more than one layer to get better results). We initialize the user and item embeddings using $\mathcal{N}(\mu = 0, \sigma = 0.01)$, and initialize the edge-weights layers using *He normal initializer* (e.g. $\boldsymbol{w}^{(o)}$, $\boldsymbol{w}_e$, $\boldsymbol{w}_u$ in Eq. (5.6), (5.18), (5.21), respectively). In the four datasets used in general recommendation task (e.g ML-100k, ML-1M, Netflix, Epinions), to avoid too many *zero paddings* for users with a smaller number of consumed items or too many context items are kept in the memory, which unnecessarily slow down the model's execution, we follow [129] to limit the number of context items using latest *s* consumed items. We search s in {5, 10, 20}. In the two shopping basket-based recommendation datasets (i.e. IJCAI-15 and TaFeng), since the maximum number of items in a transaction is small (e.g. 13 in IJCAI-15, and 18 in TaFeng), we consider all the other items in the same transaction with the target item as its context items. All the hyper-parameters are tuned using the development dataset. Our source code is available at: *https://github.com/thanhdtran/SDMR*.

Evaluation Metrics: We evaluate all models' performance by two widely used metrics: Hit Ratio (*HIT*@k), and Normalized Discounted Cumulative Gain (NDCG@k), where k is a truncated number or *top-k* item recommendation. Intuitively, *HIT*@k shows whether the test item is in the *top-k* list or not, while *NDCG*@k accounts for the position of the hits by assigning higher scores to the hits at top ranks and downgrading the scores

Table 5.3: General Recommendation Task: Overall performance of the baselines, and our proposed SDP, SDM, and SDMR on four datasets. The last four lines show the relative improvement of the SDM and SDMR over the best baseline method in General Recommenders (Group 1) and Sequential Recommenders (Group 2), respectively.

Method type	Method	**ML-100k**		**ML-1M**		**Netflix**		**Epinions**	
		HIT@10	NDCG@10	*HIT*@10	NDCG@10	*HIT*@10	NDCG@10	*HIT*@10	NDCG@10
General Recommenders (Group 1)	Item-KNN	0.166	0.073	0.235	0.110	0.039	0.019	0.121	0.096
	SLIM	0.520	0.298	0.677	0.420	0.358	0.212	0.249	0.189
	MF-BPR	0.554	0.316	0.595	0.352	0.352	0.193	0.384	0.232
	CML	0.596	0.326	0.662	0.390	0.447	0.254	0.376	0.237
	NeuMF++	0.623	0.341	0.716	0.438	0.509	0.279	0.428	0.274
	CMN++	0.620	0.344	0.729	0.442	0.523	0.293	0.423	0.272
Sequential Recommenders (Group 2)	PRME	0.638	0.381	0.724	0.486	0.509	0.329	0.538	0.346
	PRME_s	0.674	0.398	0.734	0.491	0.539	0.348	0.380	0.244
	TransRec	0.684	0.402	0.770	0.524	0.511	0.345	0.551	0.357
	Caser	0.674	0.386	**0.826**	0.606	0.480	0.253	0.326	0.268
Ours	SDP	0.616	0.349	0.694	0.424	0.497	0.279	0.416	0.266
	SDM	**0.713**	0.435	**0.816**	0.584	**0.584**	0.379	**0.575**	0.390
	SDMR	**0.695**	**0.562**	**0.810**	**0.662**	**0.592**	**0.449**	**0.568**	**0.423**
Compared to Group 1	Imprv. of SDM	14.54%	26.51%	11.93%	32.13%	11.71%	29.32%	34.35%	42.34%
	Imprv. of SDMR	11.65%	63.44%	11.11%	49.77%	13.24%	53.20%	32.71%	54.38%
Compared to Group 2	Imprv. of SDM	4.24%	8.21%	-1.21%	-3.63%	8.35%	8.91%	4.36%	9.24%
	Imprv. of SDMR	1.61%	39.80%	-1.94%	9.24%	9.83%	29.02%	3.09%	18.49%

to hits by log_2 at lower ranks.

5.2.4 Experimental Results

RQ1: Overall results in general recommendation task: The performance of our proposed models and the baselines are shown in Table 5.3. First, we observe that SDP significantly outperformed BPR in all four datasets in *Group-1 datasets*, improving *HIT*@10 from 8.33$\sim$41.19%, and NDCG@10 from 10.44$\sim$44.56%. Even though SDP and BPR shared the same loss function, the difference between them is SDP measured a signed distance score between a target user and a target item via a MLP which modeled a non-linear interaction between them, while BPR went along with Matrix Factorization that exploited inner product. This result confirms the effectiveness of using signed distance based similarity over inner product in the general recommendation task. Second, we compare SDP with CML. CML worked by trying to minimize the squared Euclidean distance scores between target users and target items. Our SDP, in another hand, works by minimizing signed distance scores of non-linear interactions (via non-linear activation functions) between target users and target items. We observe that SDP performed better than CML in all *Group-1 datasets*, improving *HIT*@10 from 8.33∼11.19%, and NDCG@10 from 7.06∼12.24%. On average, SDP improved *HIT*@10 by 7.5% and *NDCG*@10 by 9.5% compared to CML. Our SDP even gain competitive results compared to NeuMF++ and CMN++. On average, SDP is just slightly worse than NeuMF++ and CMN++ by -2.67% for *HIT*@10, and -1.68% for *NDCG*@10. All of these results show the effectiveness of using signed distance in our SDP model.

Next, we compare SDM with neighborhood-based baselines. Both SLIM and item-KNN used previously consumed items of a user to make the prediction for the next item. SDM significantly outperformed both baselines, improving *HIT*@10 from 20.53∼130.92% and NDCG@10 from 39.05∼106.35% compared with SLIM. It is an obvious result because the neighborhood-based baselines barely measured linear similarities between the target item and the user's consumed items. In contrast, our SDM produced signed distance scores and assigned personalized metric-based attention weights to each of consumed items that contribute to the target item.

We then compare SDM with CMN++ and NeuMF++. SDM outperformed CMN++ in all *Group-1 datasets*, improving *HIT*@10 from 11.71∼35.93% and NDCG@10 from 26.51∼43.38%. On average, it improves *HIT*@10 by 18.63% and *NDCG@10* by 32.84% compared to CMN++. This result shows the effectiveness of our personalized metric-based attention with signed distance and item-based neighborhood design over the traditional inner product-based attention in a user-based neighborhood design in CMN++. SDM also outperformed NeuMF++, improving *HIT*@10 from 13.97∼34.35%, and NDCG-@10 from 27.42∼42.34%. On average, in all *Group-1 datasets*, SDM outperformed all the baselines in "General Recommenders" (Group 1), improved *HIT*@10 by 18.13% and NDCG@10 by 32.58% compared to the best baseline in *Group 1*.

Table 5.4: Shopping basket-based Recommendation Task: Overall performance of the baselines, and our proposed models on two datasets. The last two lines show the relative improvement of the SDM and SDMR over the best baseline.

Method	**IJCAI-15**		**Ta-Feng**	
	HIT@10	NDCG@10	*HIT*@10	NDCG@10
PRME	0.276	0.177	0.594	0.365
PRME_s	0.229	0.133	0.590	0.355
TransRec	0.262	0.168	0.622	0.401
Caser	0.173	0.096	0.605	0.373
SDP	0.323	0.201	0.633	0.401
SDM	0.316	0.189	**0.646**	0.439
SDMR	**0.336**	**0.222**	0.627	**0.559**
Imprv. of SDM	14.49%	6.78%	3.86%	9.48%
Imprv. of SDMR	21.74%	25.42%	0.80%	39.40%

Finally, we look at the performance of SDMR model, which is the proposed fusion of SDP and SDM. Compared to SDM, our SDMR insignificantly downgrades SDM on *HIT*@10 measurement with a very small amount, but it does help a lot in refining the ranking of items and boosting *NDCG*@10 results. As shown in Table 5.3, SDMR improved from 8.46∼29.20% for *NDCG*@10, and by 17.37% for *NDCG*@10 on average compared to SDM in *Group-1 datasets*. SDMR also surpassed all the methods in *Group 1*. On average, SDMR improved *HIT*@10 by 17.18% and NDCG@10 by 55.20% compared to the best model in *Group 1*.

We also compared our models with some strong sequential models in Table 5.3. Sequential models exploited consuming time of items and model their rigid orders, which often lead to a much improved performance compared to general recommendation models in *Group-1 baselines*. As such, compared to the best sequential baseline model, on average, SDM improves *HIT*@10 by 3.94% and *NDCG*@10 by 5.68% , and SDMR improves *HIT*@10 by 3.15% and *NDCG*@10 by 24.14% compared to the best sequential model reported in Table 5.3.

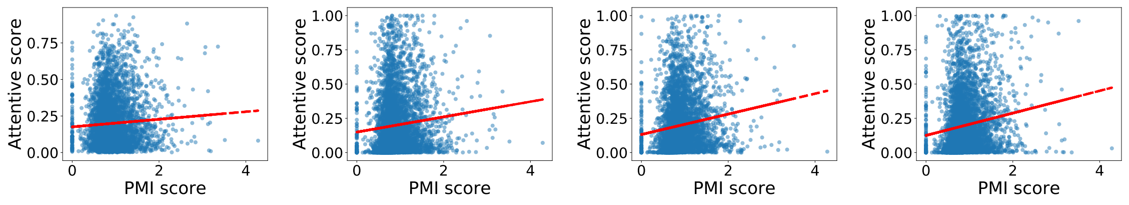
Overall results in shopping basket-based recommendation task: Table 5.4 shows the performance of our models and sequential baselines in *Group-2 datasets*. Again, our models outperformed all the sequential baselines. On average, SDM improved *HIT*@10 by 9.2% and *NDCG*@10 by 8.1%, SDMR improved *HIT*@10 by 11.3% and *NDCG*@10 by 32.4% compared to the best reported baseline.

RQ2: Understanding our multi-hop personalized metric-based attention design? In the previous section, we see that our models outperformed many strong baselines in **six** different datasets of the **two** different recommendation problems. In this part, we explore why did we achieve those better results? As "attention is all you need" [113], the core reason brought us an surpassed performance accredit to the metric-based attention which are further refined via multi-hop design. Therefore, we want to explore quantitatively and qualitatively how our attention with multi-hop design worked by answering two smaller research questions: (i) what did our metric-based attention with multi-hop design learn?, (ii) did the metric-based attention with multi-hop design improve recommendation results? Without a special mention, since our SDMR model just learned a combination between SDP and SDM without re-learning the learned-already parameters in SDP and SDM, we explore SDM in this section to understand how attention with multi-hop design works. Note that we conduct this analysis for ML-100k only due to space limitation and the availability of movies genre in ML-100k (for visualization in Figure 5.8).

What did our metric-based attention with multi-hop design learn? To answer this research question, we first measure the *point-wise mutual information* (PMI) between two certain items j and k as:

$$PMI(j,k) = log\frac{P(j,k)}{P(j) \times P(k)} \tag{5.23}$$

where $P(j,k)$ is the joint probability between two items j and k, which shows how likely j and k are co-preferred ($P(j,k) = \frac{\#(j,k)}{|D|}$, where D denotes a collection of all item-item pairs, and $|D|$ refers to the total number of item-item co-occurrence pairs in D). Similarly,



(a) Hop 1. (b) Hop 2. (c) Hop 3. (d) Hop 4.

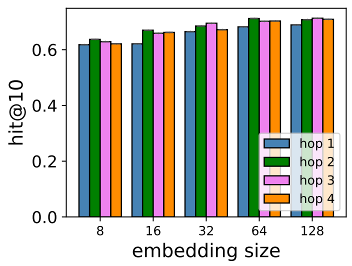
Figure 5.6: ML-100K: Scatter plots of PMI scores and attentive scores generated by SDM with h hops (h={1, 2, 3, 4} from left to right). The red lines are the linear trend lines. The Pearson correlation between two scores increases when h increased.

$P(j)$ and $P(k)$ are the probabilities of the item j and k appears in D, respectively (e.g. $P(j) = \frac{\#(j)}{|D|}$, $P(k) = \frac{\#(k)}{|D|}$). Intuitively, a PMI score between two items shows how likely the two items are co-purchased/co-preferred. The higher the PMI score between j and k is, the more likely the user will purchase j if k was purchased before.

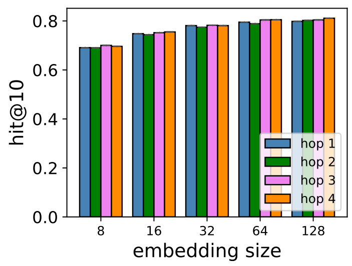
We denote SDM-h is the SDM model with h hops. Now, given a target item j and the user's context items k, *SDM-h* will assign attentive scores for all (j, k) pairs. We also get PMI scores (from Eq. (5.23)) of (j, k) pairs. Next, we plot a scatter plot of PMI scores and attentive scores for all (j, k) pairs to see the relationship between the two scores. Our results for ML-100k dataset is shown in Figure 5.6.

In Figure 5.6, the Pearson correlation between PMI scores and attentive scores are 0.059, 0.097, 0.143, and 0.146 for SDM-1, SDM-2, SDM-3 SDM-4, respectively. It indicates that as we increase the number of hops in SDM model, PMI scores and attentive scores are more positively correlated. In another word, as we increase number of hops, our metric-based attention with multi-hop design will assign higher weights for co-purchased items, which is what we desire.

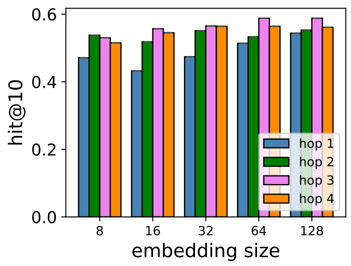
Furthermore, scatter plots in Figure 5.6(a) presents that there is a high density of points with small attentive scores. This indicates that attention in SDM-1 is distributed to several items (which is somewhat close to equally focusing on context items). However, when we increase the number of hops h, the density spreads up to the top, indicating that the



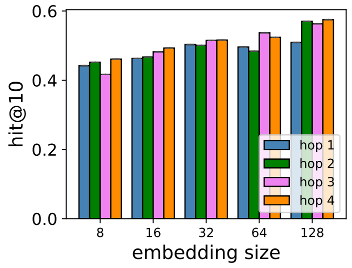
(a) ML-100K.



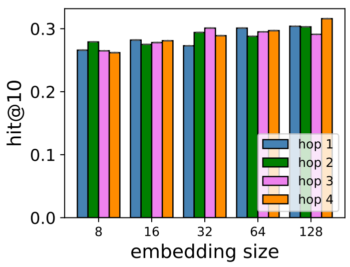
(b) ML-1M.



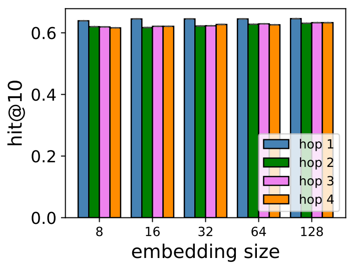
(c) Netflix.



(d) Epinions.



(e) IJCAI-15



(f) TaFeng.

Figure 5.7: Comparison of varying the number of hops regarding different embeddings sizes in the six datasets.

model tends to give a higher attention to some context items, which can be more relevant than others. This observation is consistent with "learning to attend" in [127, 128].

Did the metric-based attention with multi-hop design improve recommendation results? We answer this research question by showing the results of SDM model when varying number of hops h from $\{1, 2, 3, 4\}$ with different embedding sizes and visualize attention scores of SDM-h with a random observation as follows:

Varying number of hops with different embedding sizes: The performance of SDM-h regarding $HIT@10$ with h from $\{1, 2, 3, 4\}$ and embedding size from $\{8, 16, 32, 64, 128\}$ is presented in Figure 5.7. We see that more hops tend to give additional improvement in all 6 datasets, except in Tafeng dataset where SDM with more hops overfitted. In ML-100k and ML-1M, the optimal number of hops are 3 or 4. In Netflix, SDM with 3 hops performed well. In Epinions and IJCAI-15, SDM-4 tends to achieve better results. Overall, the selection of the number of hops depends on the dataset complexity,

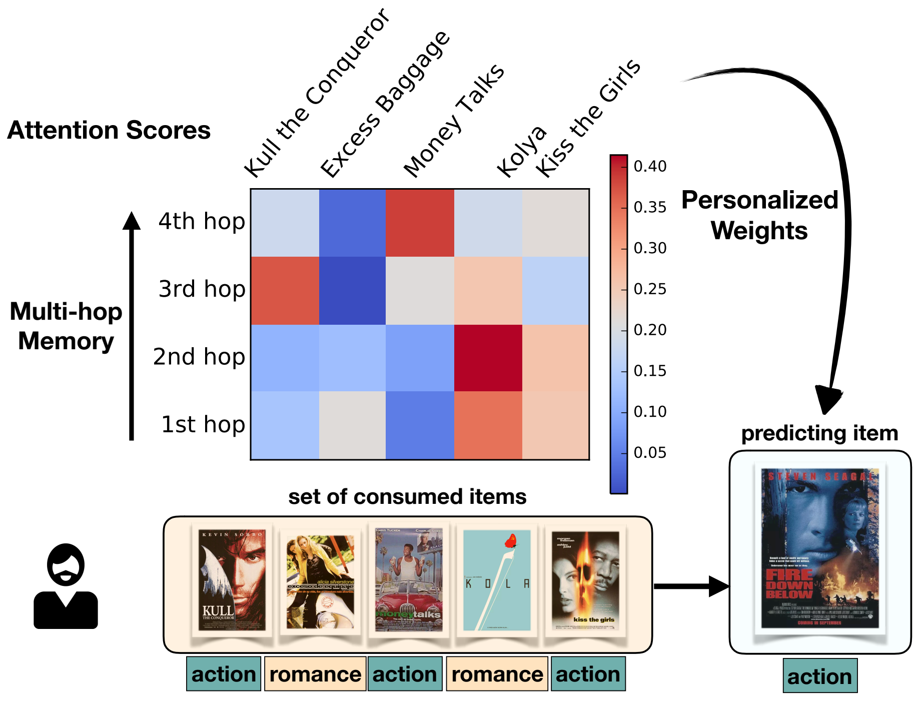


Figure 5.8: Multi-hop Attention visualization.

and it varies from datasets to datasets.

Attention Visualization: Lastly, to visualize how the personalized metric-based attention with multi-hop design works, we chose one user from ML-100K data. The learned weights at each hop of SDM is shown in Figure 5.8. The target item in this example is an action movie called *Fire Down Below* (1997). The first two hops of SDM assigned high weights to two romance movies, and the lowest score to the action movie *Money Talks* (1997). The 3rd-hop and 4th-hop attention refined the weights of movies to better reflect the correlations and similarities *w.r.t* the target movie. At last, *Money Talks* (1997) was assigned with the highest weight 0.386, and the total weights of two romance movies decreased to less than 0.2. This result shows the effectiveness of our multi-hop SDM model.

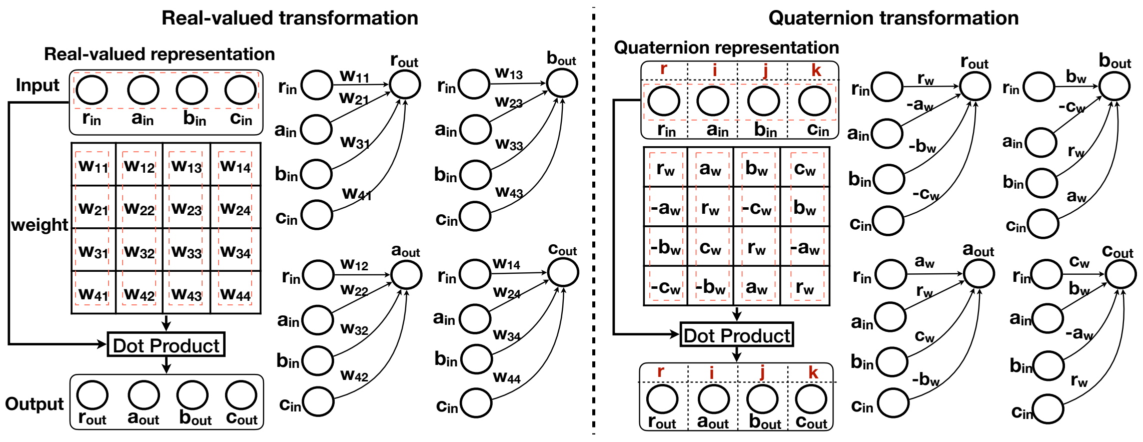

Real-valued transformation
Real-valued representation
Input
weight
Dot Product
Output
Quaternion transformation
Quaternion representation
r i j k
Dot Product

Figure 5.9: Comparison between real-valued transformation (Left) and Quaternion transformation (Right). We replace Hamilton product in Quaternion space with an equivalent dot product in real space for an easy reference.

5.3 Recommending Products with a Quaternion Representation Based Approach

In this section, we propose a Quaternion-based neural recommender system that models both long-term and short-term user preferences. Unlike the prior works [48, 49] which rely on Euclidean space, our proposed recommender system models both user's long-term and short-term preferences in a hypercomplex system (i.e., Quaternion Space) to further improve the recommendation quality.

Concretely, we utilize Quaternion representations for all users, items and neural transformations in our proposed models. There are numerous benefits of the Quaternion utilization over the traditional real-valued representations in Euclidean space: (1) Quaternion numbers/vectors consist of a real component and three imaginary components (i.e. $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$), encouraging a richer extent of expressiveness; (2) instead of using dot product in Euclidean space, Quaternion numbers/vectors operate on Hamilton product, which matches across multiple (inter-latent) Quaternion components and strengthens their inter-latent interactions, leading to a higher expressive model; (3) the weight sharing nature of

Hamilton product leads to a model with a smaller number of parameters.

To illustrate these benefits of the Quaternion utilization, we show a comparison of a transformation process with Quaternion representations *vs.* real-valued representations in Figure 5.9. In Euclidean space, different output dimensions are produced by multiplying the same input with different weights. Given a real-valued 4-dimensional vector $[r_{in}, a_{in}, b_{in}, c_{in}]$, it takes a total of 16 parameters (i.e. 16 degrees of freedom) to transform into $[r_{out}, a_{out}, b_{out}, c_{out}]$. For Quaternion transformation, the input vector now is represented with 4 components, where r_{in} is the value of the real component, a_{in}, b_{in}, c_{in} are the corresponding values of the three imaginary parts $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$. Due to the weight sharing nature of Hamilton product (refer to the Equa (5.26) in Section 5.3.2), different output dimensions take different combinations of the same input with only 4 weighting parameters $\{r_w, a_w, b_w, c_w\}$. The Quaternions provide a better inter-dependencies interaction coding and reduce 75% of the number of parameters compared with real-valued representations in Euclidean space (e.g., 4 unique parameters vs. 16 parameters).

To our best of knowledge, we are the first work that fully utilizes Quaternion space in modeling both user's long-term and short term interests. Furthermore, to increase our model's robustness, we propose a Quaternion-based Adversarial attack on Bayesian Personalized Ranking (QABPR) loss. As far as we know, we are the first, applying adversarial attack on Quaternion representations in the recommendation domain.

We first describe the problem definition and the preliminary on Quaternion representations. Then, we detail our Quaternion based recommenders and present the experimental settings and experimental results.

5.3.1 Problem Definition

Denote U=$\{u_1, u_2, ..., u_m\}$ as a set of all users where $m = |U|$ is the total number of users, and P=$\{p_1, p_2, ..., p_n\}$ as a set of all items where $n = |P|$ is the total number of items. Bold versions of those variables, which we will introduce in the following sections, indicate their respective latent representations/embeddings. Each user $u_i \in U$ consumes items in P, denoted by a chronological list $T^{(u_i)}$. We denote $L^{(u_i)}$ as the chronological list of long-term consumed items of u_i, and $S^{(u_i)}$ as the chronological list of short-term consumed items of u_i (i.e. s most recently consumed items in chronological order of the user u_i), $L^{u_i} \cup S^{u_i} = T^{(u_i)}$. Note that bold versions of $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are used to indicate the three imaginary parts of a Quaternion, while their subscript versions are used as indices.

In this work, we propose and build Quaternion-based recommender systems by using both long-term and short-term user interests, denoted as $P(p_j|L^{(u_i)}, S^{(u_i)})$. Under an assumption that $L^{(u_i)}$ and $S^{(u_i)}$ are independent given the target item p_j, we model $P(p_j|L^{(u_i)}, S^{(u_i)})$ by modeling the user's long-term interest $P(p_j|L^{(u_i)})$ and short-term interest $P(p_j|S^{(u_i)})$ separately by using two different Quaternion-based neural networks. Then, we automatically fuse the two models to build a more effective recommender system.

5.3.2 Preliminary

In this section, we cover important background on Quaternion Algebra and Quaternion Operators that we use to design our models.

Quaternion number: In mathematics, Quaternions are a hypercomplex number system. A Quaternion number *X* in a Quaternion space $\mathbb{H}$ is formed by a real component (*r*) and three imaginary components as follows:

$$X = r + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}, \tag{5.24}$$

where $\boldsymbol{ijk} = \boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$. The non-commutative multiplication rules of quaternion numbers are: $\boldsymbol{ij} = \boldsymbol{k}$, $\boldsymbol{jk} = \boldsymbol{i}$, $\boldsymbol{ki} = \boldsymbol{j}$, $\boldsymbol{ji} = -\boldsymbol{k}$, $\boldsymbol{kj} = -\boldsymbol{i}$, $\boldsymbol{ik} = -\boldsymbol{j}$. In

Equa (5.24), r, a, b, c are real numbers $\in \mathbb{R}$. Note that we can extend r, a, b, c to real-valued vectors to obtain Quaternion embeddings, which we use to represent users/items' latent features and conduct neural transformations. Operations on Quaternion embeddings are similar to Quaternion numbers.

Component-wise Quaternion Operators: Let f define an algebraic operator in real space $\mathbb{R}$. The *component-wise Quaternion operator* f on two Quaternions $X, Y \in \mathbb{H}$ is defined as:

$$f(X, Y) = f(r_X, r_Y) + f(a_X, a_Y)\boldsymbol{i} + f(b_X, b_Y)\boldsymbol{j} + f(c_X, c_Y)\boldsymbol{k} \tag{5.25}$$

For instance, if f is an *addition* operator (i.e. $f(a, b) = a + b$), then $f(X, Y)$ returns a component-wise Quaternion addition between X and Y. If f is a *dot product* operator (i.e. $f(a, b) = a^T b$), then $f(X, Y)$ returns a component-wise Quaternion *dot product* between X and Y. A similar description is applied when f is either *subtraction*, *scalar multiplication*, *product*, *softmax*, or *concatenate* operator, *.etc*.

Hamilton Product: The Hamilton product (denoted by the $\otimes$ symbol) of two Quaternions $X \in \mathbb{H}$ and $Y \in \mathbb{H}$ is defined as:

$$\begin{aligned} X \otimes Y = &(r_X r_Y - a_X a_Y - b_X b_Y - c_X c_Y) + \\ &(r_X a_Y + a_X r_Y + b_X c_Y - c_X b_Y)\boldsymbol{i} + \\ &(r_x b_Y - a_X c_Y + b_X r_Y + c_X a_Y)\boldsymbol{j} + \\ &(r_X c_Y + a_X b_Y - b_X a_Y + c_X r_Y)\boldsymbol{k} \end{aligned} \tag{5.26}$$

Activation function on Quaternions: Similar to [134, 135], we use a *split activation function* because of its stability and simplicity. *Split activation function* β on a Quaternion X is defined as:

$$\beta(X) = \alpha(r) + \alpha(a)\boldsymbol{i} + \alpha(b)\boldsymbol{j} + \alpha(c)\boldsymbol{k} \tag{5.27}$$

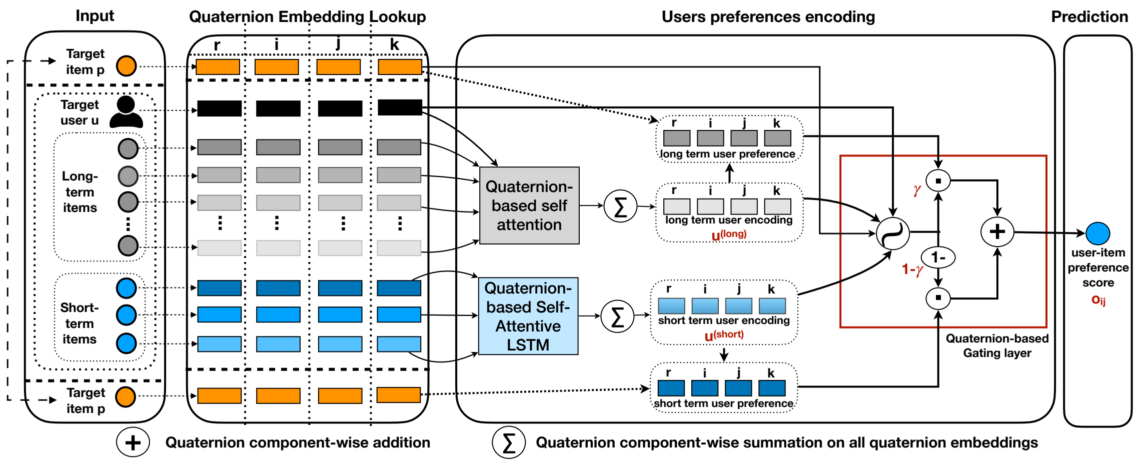

Input
Quaternion Embedding Lookup
Users preferences encoding
Prediction
Target item p
Target user u
Long-term items
Short-term items
Quaternion-based self attention
Quaternion-based Self-Attentive LSTM
long term user preference
long term user encoding
$u^{(long)}$
short term user encoding
$u^{(short)}$
short term user preference
Quaternion-based Gating layer
user-item preference score
o_{ij}
Quaternion component-wise addition
Quaternion component-wise summation on all quaternion embeddings

Figure 5.10: Our proposed architecture for modeling both long and short-term user interests using Quaternion representations.

, where α is any standard activation function for real values.

Concatenate four components of a Quaternion: concatenates all four Quaternion components into one real-valued vector:

$$[X] = [r_X, a_X, b_X, c_X] \tag{5.28}$$

5.3.3 Method

Figure 5.10 shows an overview of our proposals. First, our QUaternion-based self-Attentive Long term user Encoding (*QUALE*) learns a user's long-term interest by using long-term consumed items and the target item. Second, our QUaternion-based self-Attentive Short term user Encoding (*QUASE*) encodes the user's short-term intent by using short-term consumed items and the target item. Then our QUaternion-based self-Attentive Long Short term user Encoding (*QUALSE*) fuses both of the user preferences by using a Quaternion-based gating layer. We describe each component as follows:

QUaternion-based self-Attentive Long term user encoding (our QUALE model:)

The most widely used technique for modeling the user long-term interests is the Asymmetric-SVD (ASVD) [35] model. Its basic idea is to encode each user and item by latent representations where the user representation is encoded by summing latent representations of the user's interacted items. To an extent, we propose a QUaternion-based self-Attentive Long term user Encoding (*QUALE*). *QUALE* represents each user and each item as Quaternion embeddings. Then, we encode each user by attentively summing Quaternion embeddings of her interacted items as follows:

$$\boldsymbol{u_i}^{(long)} = \sum_{k=1}^{|L^{(u_i)}|} \alpha_k \times \boldsymbol{p_k}^{(long)} \tag{5.29}$$

where $\boldsymbol{u_i}^{(long)}, \boldsymbol{p_k}^{(long)} \in \mathbb{H}$. The summation "$\sum$" and the multiplication "$\times$" are Quaternion component-wise operators, which are calculated by using Equa (5.25). We use our proposed Quaternion personalized self-attention mechanism to assign attentive scores $\alpha_k \in \mathbb{H}$ for different long-term items p_k.

Our *QUALE* model has four layers: Input, Quaternion Embedding, Encoding, and Output layers. We detail each layer as follows:

Input: *QUALE* requires a target user u_i, a target item p_j, and the user's list of l long-term items $L^{(u_i)}$ with $|L^{(u_i)}| = l$. l could be simply set to the maximum number of long-term items among all the users in a dataset. However, we observed that only several users in our datasets consumed an extremely large number of items compared to the majority of users. Hence, we set l to the upper bound of the boxplot approach (i.e. Q3 + 1.5IQR, where Q3 is the *third quartile*, and IQR is the *Interquartile range* of the sequence length distribution of all users). If a user has consumed less than l items, we pad the list with *zeroes* until its length reaches l.

Quaternion Embedding layer: It holds two Quaternion embedding matrices: a user context Quaternion embedding matrix $\mathcal{U}^{(long)} \in \mathbb{H}^{m \times d}$, and an item Quaternion embedding matrix $\mathcal{P}^{(long)} \in \mathbb{H}^{n \times d}$. Here, m and n are the respective number of users and items in the system. d is the Quaternion embedding size, and is measured by the total size of real-valued vectors of four Quaternion components ($d = |r| + |a| + |b| + |c|$, and $|r| = |a| = |b| = |c| = d/4$). By passing the target user u_i, the target item p_j, and long-term items p_k in the *Input* layer through the two respective Quaternion embedding matrices, we obtain the corresponding user context Quaternion embedding $\boldsymbol{q_i}^{(long)}$, target item Quaternion embedding $\boldsymbol{p_j}^{(long)}$ and long-term item Quaternion embeddings $\boldsymbol{p_k}^{(long)}$.

Encoding layer: Its main goal is to compute attentive scores for l Quaternion item embeddings in Equa (5.29). To do so, we propose a Quaternion personalized self-attention mechanism as follows:

We first compute the Hamilton product between each long-term item Quaternion embedding $\boldsymbol{p_k}^{(long)}$ ($k = \overline{1, l}$) and the Quaternion context embedding $\boldsymbol{q_i}^{(long)}$ of the target user u_i. Next, we use Equa (5.25) to multiply the results with the scaling factor $1/\sqrt{d}$ to eliminate the scaling effects. Then, we apply Component-wise Softmax (Equa (5.25)) to obtain Quaternion attention scores as follows:

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_l \end{bmatrix} = \text{ComponentSoftmax}\left(\begin{bmatrix} \boldsymbol{p_1}^{(long)} \otimes \boldsymbol{q_i}^{(long)}/\sqrt{d} \\ \boldsymbol{p_2}^{(long)} \otimes \boldsymbol{q_i}^{(long)}/\sqrt{d} \\ \dots \\ \boldsymbol{p_l}^{(long)} \otimes \boldsymbol{q_i}^{(long)}/\sqrt{d} \end{bmatrix}\right) \tag{5.30}$$

.

To obtain the attentive long-term user encoding $\boldsymbol{u_i}^{(long)}$ of the user u_i, we first perform the component-wise product between the attention scores $[\alpha_1, \alpha_2, ..., \alpha_l]$ obtained in Equa (5.30) with its corresponding item Quaternion embeddings $[p_1^{(long)}, p_2^{(long)}, ..., p_l^{(long)}]$. Then we sum them up to obtain $\boldsymbol{u_i}^{(long)}$ as follows:

$$\boldsymbol{u_i}^{(long)} = \sum_{k=1}^{l} \alpha_k \times \boldsymbol{p_k}^{(long)} \tag{5.31}$$

Our proposed Quaternion personalized self-attention mechanism vs. the existing

self-attention mechanism: Our proposed Quaternion personalized self-attention mechanism is different from the self-attention mechanism that has been widely used in the NLP tasks in two aspects. First, unlike the prior work [136], which uses a single global context to assign attentive scores for different dialogue states, our attention mechanism provides personalized contexts for different users. In the recommendation domain, the long-term/general user interests are supposed to be changed slowly, but user interests are various across users. In other words, a user's long-term context is quite static, but different from another user. Hence, using personalized contexts for different users is better than using a single global context, which is not personalized. Second, our attention mechanism adopts Hamilton product and works for Quaternion embeddings as input, instead of the real-valued embeddings like traditional self-attention mechanisms.

Output: We produce a long-term preference score $o_{ij}^{(long)}$ between the target user u_i and the target item p_j by computing the Component-wise dot product between the user long-term Quaternion encoding $\boldsymbol{u_i}^{(long)}$ obtained in Equa (5.31) and the target item Quaternion embedding $\boldsymbol{p_j}^{(long)}$. This results in a Quaternion score . To obtain a real-valued scalar preference score used in the parameter estimation phase, we compute the average of the scalar values of four Quaternion components by following [137]:

$$o_{ij}^{(\text{long})} = \text{Average}(\text{ComponentDot}(\boldsymbol{u_i}^{(long)}, \boldsymbol{p_j}^{(long)})) \tag{5.32}$$

QUaternion-based self-Attentive Short term user Encoding (our QUASE model):

RNN-based models have gained a lot of attention because of their capability to capture item-to-item relationships [44, 48, 138]. However, due to its limitation in modeling a long sequence, we only exploit the RNN architecture to encode a user's short-term interest. Recently, [134] has introduced a Quaternion LSTM (QLSTM) model and has shown its efficiency and effectiveness over a traditional real-valued LSTM model. However, QLSTM used only the last hidden state as a latent summary of the input, which is suboptimal. To an extent, we propose a Quaternion-based self-Attentive LSTM model to learn a user's short-term interest. We name our proposal as a QUaternion-based self-Attentive Short term user Encoding (QUASE). *QUASE* has 4 layers: Input, Quaternion Embedding, Encoding, and Output layers. We describe each layer as follows:

Input: A target item p_j, and the chronological list of s short-term consumed items $S^{(u_i)}$ of the target user u_i with $|S^{(u_i)}| = s$, where s represents the maximum number of short-term items among all the users in a dataset. If a user has consumed less than s items, we pad the list with *zeroes* until its length reaches s.

Quaternion Embedding layer: It holds an item Quaternion Embedding matrix $\mathcal{P}^{(short)} \in \mathbb{H}^{n\times d}$. By passing the target item p_j, and s short-term items in the $S^{(u_i)}$ of the target user u_i through P^{short}, we obtain their corresponding Quaternion embeddings $\boldsymbol{p_j}^{(short)}$, and $\{\boldsymbol{p_1}^{(short)}, \boldsymbol{p_2}^{(short)}, ..., \boldsymbol{p_s}^{(short)}\}$.

Encoding layer: In this layer, we adapt the recently introduced Quaternion-based LSTM to model the item-item sequential transition. Denote $\boldsymbol{p_t^{(short)}}$ is the Quaternion embedding of the t^{th} short-term item $p_t \in S^{(u_i)}$ $(t = \overline{1, s})$. Let f_t, i_t, o_t, c_t, and h_t be the forget gate, input gate, output gate, cell state, and the hidden state of a Quaternion LSTM cell at time step t, respectively. We compute these variables as follows:

$$\begin{aligned}
f_t &= \sigma(W_f \otimes \boldsymbol{p_t}^{(short)} + R_f \otimes \boldsymbol{h_{t-1}} + g_f) \\
i_t &= \sigma(W_i \otimes \boldsymbol{p_t}^{(short)} + R_i \otimes \boldsymbol{h_{t-1}} + g_i) \\
o_t &= \sigma(W_o \otimes \boldsymbol{p_t}^{(short)} + R_o \otimes \boldsymbol{h_{t-1}} + g_o) \\
c_t &= f_t \times c_{t-1} + i_t \times \text{tanh}(W_c \otimes \boldsymbol{p_t}^{(short)} + R_c \otimes \boldsymbol{h_{t-1}} + g_c) \\
h_t &= o_t \times tanh(c_t)
\end{aligned} \tag{5.33}$$

, where $W_f, R_f, W_i, R_i, W_o, R_o, W_c, R_c$ are Quaternion weight matrices. g_f, g_i, g_o, g_c are

Quaternion bias vectors. f_t, i_t, o_t, c_t, h_t are Quaternion vectors. The "$\times$" sign denotes a component-wise *product* operator, which is calculated using Equa (5.25). *sigmoid* σ and *tanh* are split activation functions and are computed using the Equa (5.27).

Using Equa (5.33), given s short-term consumed items $p_1, p_2, ..., p_s$, we obtain their respective output Quaternion hidden states $\boldsymbol{h_1}, \boldsymbol{h_2}, ..., \boldsymbol{h_s}$. Then, we propose a Quaternion self-attention mechanism to combine all s output Quaternion hidden states before using it to predict the next item. Different from the long-term user preferences where they are supposed to be static or changed very slowly, the short-term user interests are dynamic and changed quickly. Hence, using a static user context for each user to make personalized attention like what we did for the *QUALE* model is not ideal. Instead, we define a Quaternion global context vector to capture the sequential transition patterns from item to item among all the users. Denote q as a Quaternion global context vector, the Quaternion-based self-attention score of each hidden state h_t is measured by:

$$\begin{bmatrix} \alpha_1^{(short)} \\ \alpha_2^{(short)} \\ \dots \\ \alpha_s^{(short)} \end{bmatrix} = \text{ComponentSoftmax}\left(\begin{bmatrix} \boldsymbol{h_1} \otimes \boldsymbol{q}/\sqrt{d} \\ \boldsymbol{h_2} \otimes \boldsymbol{q}/\sqrt{d} \\ \dots \\ \boldsymbol{h_t} \otimes \boldsymbol{q}/\sqrt{d} \end{bmatrix}\right) \tag{5.34}$$

, where $\alpha_1^{(short)}, \alpha_2^{(short)}, ..., \alpha_s^{(short)}$ are Quaternion numbers. To achieve the final short-term user Quaternion encoding, we perform a component-wise product between the Quaternion hidden states and their respective Quaternion attention scores, followed by a Hamilton product with a Quaternion weight matrix W and the split activation function *tanh*:

$$\boldsymbol{u_i}^{(short)} = tanh\left(W \otimes \left(\sum_{t=1}^{s} \alpha_t^{(short)} \times \boldsymbol{h_t}\right)\right) \tag{5.35}$$

Note that we also designed a Quaternion self-Attentive GRU , but its performance was slightly worse than the Quaternion self-Attentive LSTM (see Table 5.6 in Section 5.3.4).

Thus, we only described the Quaternion self-Attentive LSTM due to space limitation.

Output: Similar to Equa (5.32), we produce the user u_i short-term preference score $o_{ij}^{(short)}$ over the target item p_j as follows:

$$o_{ij}^{(short)} = Average\big(ComponentDot(\boldsymbol{u_i}^{(short)}, \boldsymbol{p_j}^{(short)})\big) \tag{5.36}$$

QUaternion-based self-Attentive Long Short term user Encoding (QUALSE): a Fusion of QUASE and QUALE models In this part, we aim to combine both user's long-term and short-term preferences modeling parts into one model, namely *QUALSE*, fusing *QUALE* and *QUASE* models. Inspired by the gated mechanism in LSTM [139] to balance the contribution of the current input and the previous hidden state, we propose a *personalized* Quaternion gated mechanism to fuse the long-term and short-term user interests learned in *QUALE* and *QUASE* models. Our *personalized* gating proposal is different to the traditional gating mechanism in two folds. First, gating weights in our proposal are in Quaternion space and the transformations are computed using the Hamilton product. Second, as users' behaviors differ from a user to another user, we additionally input the target user embeddings $\boldsymbol{u_i}$ to let the gating layer assign personalized scores for different users. The long-term and short-term interest fusion is computed as follows:

$$\begin{aligned} \gamma_{ij}^{(long)} =& \sigma\big(W_g^{(1)} \otimes [\boldsymbol{u_i}^{(long)}, \boldsymbol{u_i}^{(short)}] + W_g^{(2)} \otimes \boldsymbol{u_i} + W_g^{(3)} \otimes \boldsymbol{p_j}\big) \\ o_{ij} =& W_o^{(1)}\big[\gamma_{ij}^{(long)} \times (\boldsymbol{u_i}^{(long)} \times \boldsymbol{p_j}^{(long)})\big] + \\ & W_o^{(2)}\big[(1-\gamma_{ij}^{(long)}) \times (\boldsymbol{u_i}^{(short)} \times \boldsymbol{p_j}^{(short)})\big] \end{aligned} \tag{5.37}$$

, where $W_g^{(1)}$, $W_g^{(2)}$, and $W_g^{(3)}$ are Quaternion weight matrices, $\boldsymbol{u_i}^{(long)}$ and $\boldsymbol{u_i}^{(short)}$ are the user's long-term Quaternion encoding and short-term Quaternion encoding obtained in Equa (5.31) and (5.35), respectively. $[\cdot\, , \cdot]$ is the component-wise concatenate (Equa (5.25)) of two input Quaternion vectors. To compute the long-term gate $\gamma_{ij}^{(long)}$, $\boldsymbol{u_i}$ and $\boldsymbol{p_j}$ are introduced as an additional user context Quaternion embedding and a target item context Quaternion embedding to let the model know which long-term or short-term interests are more relevant. To measure the final output o_{ij}, since $\gamma_{ij}^{(long)}$ is a Quaternion vector while $o_{ij}^{(long)}$ and $o_{ij}^{(short)}$ are scalar values, we reconstruct the user's long-term interest by computing $\boldsymbol{u_i}^{(long)} \times \boldsymbol{p_j}^{(long)}$ and the short-term interest by measuring $\boldsymbol{u_i}^{(short)} \times \boldsymbol{p_j}^{(short)}$, which are also Quaternion vectors. Finally, to combine multiple dimensional features from the weighted long-term and short-term interest Quaternion vectors, we concatenate all their components, denoted by $[\cdot]$ (Equa (5.28)), and use two real-valued weight vectors $W_o^{(1)}$ and $W_o^{(2)}$ to produce a fused preference score as a scalar real number. Note that in *QUALSE*, *QUASE* and *QUALE* hold separated item memory to increase the their flexibility.

Parameter Estimation:

Training with Bayesian Personalized Ranking (BPR) loss: Given a Quaternion matrix $E \in \mathbb{H}^{(m+n)\times d}$ as the Quaternion embeddings of all users and items in the system, and Θ as other parameters of the model, we aim to minimize the following BPR loss function:

$$\begin{aligned}&\mathcal{L}_{BPR}(\mathcal{D}|E,\Theta)\\&= \mathop{\mathbf{argmin}}_{E,\Theta}\Bigg(-\sum_{(i,j^+,j^-)} log\sigma(o_{ij^+}-o_{ij^-})+\lambda_\Theta\|\Theta\|_2+\lambda_E\|E\|_2\Bigg)\end{aligned} \tag{5.38}$$

, where $(i,\ j^+,\ j^-)$ is a triplet of a target user, a target item, and a negative item that is randomly sampled from the items set P. $\mathcal{D}$ denotes all the training instances. o_{ij^+} and o_{ij^-} are the respective positive and negative preference scores, that are computed by Equa (5.32), (5.36), (5.37), corresponding to *QUALE*, *QUASE* and *QUALSE* models. λ_Θ and λ_E are regularization hyper-parameters.

Training with Quaternion Adversarial attacks: Previous works have shown that neural networks are vulnerable to adversarial noise [97, 107]. Therefore, to increase the robustness of our models, we propose a Quaternion Adversarial attack on BPR loss, namely *QABPR*. *QABPR* inherits from traditional adversarial attacks for computer visions [107] and recommendation systems [97] but differs from them: *QABPR* applies for Quaternion space, while the formers apply for real-valued space. To our best of knowledge, ours is the first work using adversarial training on Quaternion space in the recommendation domain.

In *QABPR*, we first define learnable Quaternion perturbation noise δ on user and item Quaternion embeddings. Then, we perform the Quaternion component-wise addition (Equa (5.25)) to obtain crafted Quaternion embeddings. The learnable Quaternion noise δ is optimized such that the model mis-ranks between positive items and negative items (i.e. negative items have higher preference scores than positive items). Particularly, a *max* player learns δ by maximizing the following cost function under the L_2 attack:

$$\begin{aligned}&\mathcal{L}_{adv}(\mathcal{D}|E^* + \delta, \Theta^*)\\&= \underset{\delta, \|\delta\|_2 \leq \epsilon}{\mathbf{argmax}} \left(- \sum_{(i,j^+,j^-)} log\sigma(o_{ij^+} - o_{ij^-}) + \lambda_\delta \|\delta\|_2 \right)\end{aligned} \tag{5.39}$$

where ϵ is a noise magnitude hyper-parameter. E^* and Θ^* are optimal values of E and Θ that are pre-learned in Equa (5.38) and are fixed in Equa (5.39). $E^* + \delta$ is the crafted Quaternion embeddings. $\lambda_\Theta \|\Theta\|_2$ and $\lambda_E \|E\|_2$ in Equa (5.38) are ignored in Equa (5.39) as they become constant terms. $\lambda_\delta \|\delta\|_2$ is the noise regularization term.

Solving Equa (5.39) is expensive. Hence, we adopt the Fast Gradient Method [107] to approximate δ as follows:

$$\delta = \epsilon \frac{\nabla_\delta \mathcal{L}_{adv}(\mathcal{D}|E^* + \delta, \Theta^*)}{\|\nabla_\delta \mathcal{L}_{adv}(\mathcal{D}|E^* + \delta, \Theta^*)\|_2} \tag{5.40}$$

Then, a *min* player aims to minimize the following cost functions that incorporate both non-adversarial and adversarial examples:

$$\begin{aligned}&\mathcal{L}_{QABPR}(\mathcal{D}|E, E+\delta^*, \Theta)\\&= \underset{E,\Theta}{\mathbf{argmin}} \left(\mathcal{L}_{BPR}(\mathcal{D}|E, \Theta) + \lambda_{adv}\mathcal{L}_{BPR}(\mathcal{D}|E+\delta^*, \Theta)\right)\end{aligned} \tag{5.41}$$

where δ^* is the adversarial noise that is already learned in Equa (5.40), and is fixed in Equa (5.41). λ_{adv} is a hyper-parameter to balance the effect of the partial adversarial loss. Training *QABPR* now becomes playing a *minimax* game, where the *min* and *max* players play alternatively. We stop the game after a fixed number of epochs (i.e. 30 epochs) and report results based on the best *validation* performance.

Note that we name our *QUALE*, *QUASE*, and *QUALSE* trained with *QABPR* loss as *AQUALE*, *AQUASE*, and *AQUALSE* with "A" denotes "adversarial", respectively.

5.3.4 Experimental Settings

In this section, we design experiments to answer the following research questions:

- **RQ1:** How do our proposals work compared to the baselines?
- **RQ2:** How do a user's long-term, short-term preference encoding models and the fused model perform?
- **RQ3:** Is using Quaternion representation helpful and why?
- **RQ4:** Are the gating fusion mechanism and the Quaternion BPR adversarial training helpful?

Datasets: We evaluate all models on **six** public benchmark datasets collected from two real world systems as follows:

- **Amazon datasets [88]**: As top-level product categories on Amazon are treated as independent datasets [47], we use 5 different Amazon category datasets to vary the sparsity, variability, and data size: *Apps for Android*, *Cellphone Accessories*, *Pet Sup-*

Table 5.5: Datasets' statistics with # of long-term items l.

Dataset	# of users	# of items	# of actions (density %)	l
Toys Games	36k	55k	251k (0.013%)	1,112
Cellphone Accessories	47k	45k	262k (0.012%)	109
Pet Supplies	25k	23k	160k (0.027%)	176
Video Games	24k	20k	196k (0.040%)	856
Apps for Android	79k	18k	555k (0.038%)	478
Yelp	22k	21k	481k (0.104%)	930

plies, *Toys and Games*, and *Video Games*.

- **Yelp dataset**: This is a user rating dataset on businesses. We use the dataset obtained from [74].

For data preprocessing, we adopted a popular *k-core* preprocessing step [88] (with k=5), filtering out users and items with less than 5 interactions. All observed ratings are considered as positive interactions and the remaining as negative interactions. The maximum number of short-term items is set to $s = 5$ in all datasets as it covers the short-term peak (see Figure 5.1). Table 5.5 summarizes the statistics of all datasets, as well as their number of long-term items l.

State-of-the-art Baselines: We compared our proposed models with **11** strong state-of-the-art recommendation models as follows:

- **AASVD**: It is an attentive version of the well-known Asymmetric SVD model (ASVD) [35], where real-valued self-attention is applied to measure attentive contribution of previously consumed items by a user.
- **QCF** [137]: It is a state-of-the-art recommender that represents users/items by Quaternion embeddings.
- **NeuMF++** [26]: It models non-linear user-item interactions by using a MLP and a Generalized MF (GMF) component. We pretrained MLP and GMF to obtain NeuMF's best performance.

- **NAIS** [37]: It is an extension of *ASVD* where contribution of consumed items to the target item is attentively assigned. We adopt $NAIS_{prod}$ version as it led to its best results.
- **FPMC** [40]: It is a state-of-the-art sequential recommender. It uses the first-order Markov to model the transition between the next item and the previously consumed items.
- **AGRU**: It is an extension of the well-known GRU4Rec [43], where we use an attention mechanism to combine different hidden states. We experiment with two attention mechanisms: real-valued self-attention, and real-valued *prod attention* proposed by [37]. Then we report its best performance.
- **ALSTM**: It is a LSTM based model. Similar to AGRU, we experiment with the real-valued self-attention and the *prod attention* [37], and then report its best results.
- **Caser** [42]: It embedded a sequence of recently consumed items into an “image” in time and latent spaces, and uses convolution neural network to learn sequential patterns as local features of an image using different horizontal and vertical filters.
- **SASRec** [47]: It is a strong sequential recommender model. It uses the self-attention mechanism with a multi-head design to identify relevant items for next predictive items.
- **Sli-Rec** [49]: It uses a time-aware controller to control the state transition. Then it uses an attention-based framework to fuse a user’s long-term and short-term preferences.
- **ALSTM+AASVD**: It is our implementation that resembles the same architecture as our proposed Quaternion fusion approach, except that it uses Euclidean space instead of Quaternion space. The purpose of implementing and using it as a baseline is to present the effectiveness of our framework and Quaternion representations over the real-valued representations.

First four baselines (AASVD, QCF, NeuMF++, and NAIS) are classified as user's long-term interest encoding models. Next four baselines (FPMC, AGRU, ALSTM, and Caser) are user's short-term interest encoding models, and *SASRec, SLi-Rec, and ALSTM+AASVD* encode both user's long-term and short-term intents. Note that we performed an experiment with *DIEN* [48] (i.e. a long short-term modeling baseline) based on the authors' public source code, which produced surprisingly low results, so we omit its detailed results. We also experimented with *ASVD*, LSTM, GRU and Quaternion LSTM but do not report their results due to space limitation and their worse results. Similarly, we omit *BPR* [111] and *FISM* [36] results due to their less impressive performance.

Experimental Protocol: We adopt a well-known and practical 70/10/20 splitting proportions to divide each dataset into train/validation (or development)/test sets [31, 110]. All user-item interactions are sorted in ascending order in terms of the interaction time. Then, the first 70% of all interactions are used for training, the next 10% of all interactions are used for development, and the rest is used for testing. We follow [33, 140] to sample 1,000 unobserved items that the target user has not interacted before, and rank all her positive items with these 1,000 unobserved items for testing models.

Evaluation metrics: We evaluate the performances of all models by using two well-known metrics: *Hit* Ratio (*HIT@N*), and Normalized Discounted Cumulative Gain (*NDCG@N*). *HIT@N* measures whether all the test items are in the recommended list or not, while *NDCG@N* takes into account the position of the test items, and assigns higher scores if test items are at top-rank positions.

Hyper-parameters Settings: All models are trained with *Adam* optimizer [130]. A learning rate is chosen from $\{0.001, 0.0005\}$, and regularization hyperparameters are chosen from $\{0, 0.1, 0.001, 0.0001\}$. An embedding size d is chosen from $\{32, 48, 64, 96, 128\}$. Note that for Quaternion embeddings, each component value is a vector of size $\frac{d}{4}$. The number of epochs is 30. The batch size is 256. The number of MLP layers

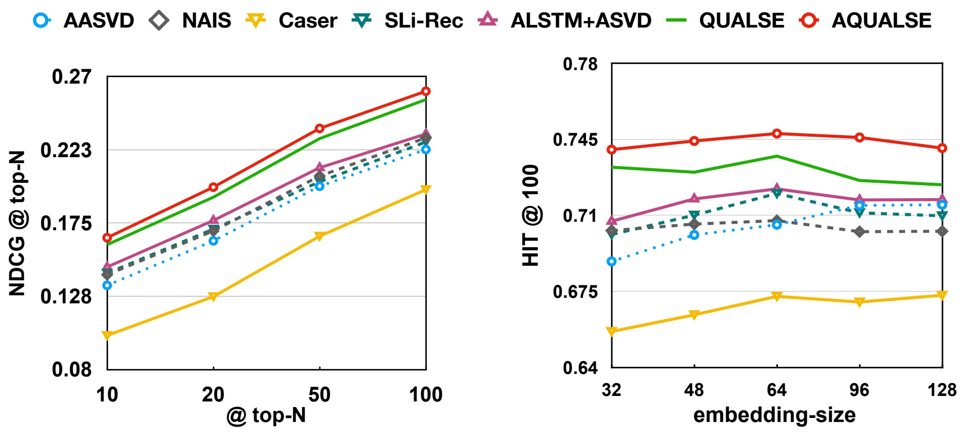
Table 5.6: HIT@100 and NDCG@100 of all models. Best performances are in *bold*, best baseline's results are underlined. The last two lines show the relative improvement of QUALSE and AQUALSE compared to the best baseline's results.

Methods	Toys Games		Cellphone Acc.		Pet Supplies		Video Games		Apps for Android		Yelp	
	HIT	NDCG	HIT	NDCG	HIT	NDCG	HIT	NDCG	HIT	NDCG	HIT	NDCG
(a) AASVD	0.4343	0.1809	0.5640	0.2443	0.5523	0.2307	0.5503	0.2229	0.7149	0.3182	0.7212	0.3580
(b) QCF	0.3869	0.1560	0.5514	0.2328	0.5319	0.2194	0.5217	0.1956	0.6638	0.2864	0.6774	0.3119
(c) NeuMF++	0.3969	0.1553	0.5467	0.2291	0.5255	0.2174	0.4944	0.1934	0.6635	0.2791	0.6810	0.3208
(d) NAIS	0.4331	0.1796	0.5648	0.2427	0.5569	0.2302	0.5587	0.2303	0.7076	0.3138	0.7277	0.3573
(e) FPMC	0.3370	0.1335	0.4805	0.1970	0.4405	0.1812	0.5065	0.1980	0.6659	0.2847	0.6704	0.3204
(f) AGRU	0.3747	0.1400	0.5211	0.2030	0.4690	0.1798	0.5337	0.1958	0.6969	0.2960	0.4722	0.1995
(g) ALSTM	0.3886	0.1419	0.5159	0.2052	0.4630	0.1685	0.5156	0.1928	0.7043	0.2883	0.5644	0.2519
(h) Caser	0.3889	0.1507	0.5747	0.2289	0.4786	0.1859	0.5502	0.1967	0.7098	0.3124	0.6718	0.3201
(i) SASRec	0.4009	0.1545	0.5579	0.2239	0.5238	0.2124	0.5472	0.2107	0.6706	0.2781	0.7193	0.3381
(j) SLi-Rec	0.4267	0.1823	0.5661	0.2387	0.5502	0.2311	0.5438	0.2276	0.7062	0.3117	0.7201	0.3516
(k) ALSTM+AASVD	0.4394	0.1864	0.5701	0.2475	0.5542	0.2326	0.5502	0.2328	0.7173	0.3207	0.7222	0.3594
Our proposals												
QUALE	0.4696	0.1997	0.6042	0.2685	0.5826	0.2483	0.5981	0.2503	0.7281	0.3248	0.7391	0.3723
QUASE (GRU)	0.4080	0.1632	0.5612	0.5807	0.2438	0.5413	0.2246	0.2207	0.7198	0.3223	0.6917	0.3324
QUASE (LSTM)	0.4095	0.1664	0.5844	0.2475	0.5453	0.2263	0.5591	0.2261	0.7300	0.3300	0.6929	0.3311
QUALSE	**0.4760**	**0.2043**	**0.6127**	**0.2777**	**0.5913**	**0.2539**	**0.6018**	**0.2551**	**0.7373**	**0.3364**	**0.7442**	**0.3781**
AQUALE	0.4831	0.2055	0.6105	0.2748	0.5902	0.2553	0.6045	0.2593	0.7346	0.3306	0.7440	0.3786
AQUASE (LSTM)	0.4495	0.1847	0.6056	0.2572	0.5520	0.2329	0.5762	0.2351	0.7285	0.3292	0.7048	0.3450
AQUALSE	**0.4921**	**0.2098**	**0.6204**	**0.2842**	**0.6011**	**0.2612**	**0.6137**	**0.2605**	**0.7477**	**0.3440**	**0.7448**	**0.3814**
Imprv. of QUALSE	**+8.33%**	**+9.60%**	**+6.61%**	**+12.20%**	**+6.18%**	**+9.16%**	**+7.71%**	**+9.58%**	**+2.79%**	**+4.90%**	**+2.27%**	**+5.20%**
Imprv. of AQUALSE	**+11.99%**	**+12.55%**	**+7.95%**	**+14.83%**	**+7.94%**	**+12.30%**	**+9.84%**	**+11.90%**	**+4.24%**	**+7.27%**	**+2.35%**	**+6.12%**

in NeuMF++ is tuned from $\{1, 2, 3\}$. The number of negative samples per one positive instance is 4 for training models. The settings of *Caser*, *NAIS*, *SASRec* are followed by their reported default settings. In training with *QABPR* loss, the regularization λ_{adv} is set to 1. The noise magnitude ϵ is chosen from $\{0.5, 1, 2\}$. The adversarial noise is added only in training process, and is initialized as *zero*. All hyper-parameters are tuned by using the validation set.

5.3.5 Experimental Results

RQ1: Performance comparison Table 5.6 shows that our proposed fused models *QUALSE* and *AQUALSE* outperformed all the compared baselines. On average, *QUALSE* improved *Hit@100* by 5.65% and *NDCG@100* by 8.44% compared to the best baseline's performances. *AQUALSE* gains additional improvement over *QUALSE*, enhancing *Hit@100* by 7.39% and *NDCG@100* by 10.83% on average compared to the best baseline. The improvement of our proposals over the baselines is significant under the Directional



(a) NDCG@topN in Video Games (b) HIT@100 in Yelp.

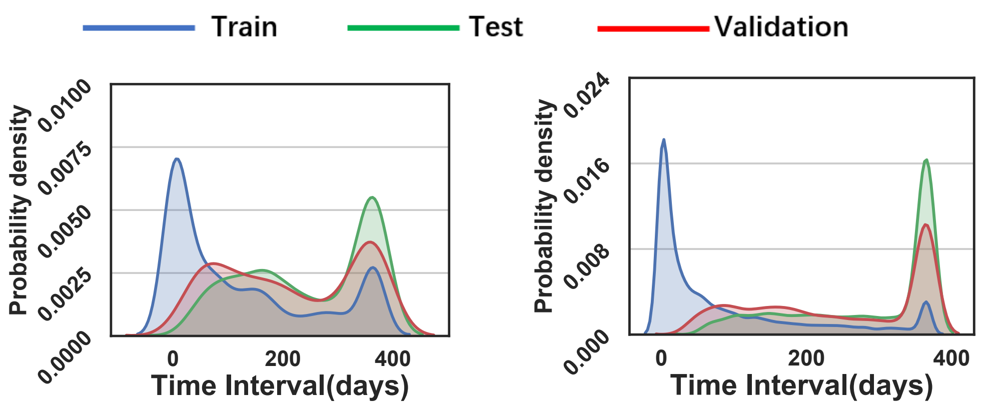
Figure 5.11: Performance of our models and the top-5 baselines when varying a *top-N* recommendation list (left) and an embedding size (right).

Wilcoxon signed-rank test (*p-value* < 0.015). We also observed similar results on all six datasets when we measure *Hit@1* and *NDCG@1*. In particular, our *QUALSE* improved *Hit@1* by 6.87% and *NDCG@1* by 8.71% on average compared with the best baseline. *AQUALSE* improved *Hit@1* by 8.43% and *NDCG@1* by 10.27% on average compared with the best baseline, confirming its consistent effectiveness.

Varying top-N recommendation list and embedding size: To further provide detailed effectiveness of our proposals, we compare QUALSE and AQUALSE models with the top-5 baselines when varying the embedding size from {32, 48, 64, 96, 128} and the *top-N* recommendation list from {10, 20, 50, 100}.

Figure 5.11(a) shows that even with small *top-N* values (e.g., @10), our models consistently outperformed all the compared baselines in the *Video Games* dataset, improving the ranking performance by a large margin of 9.25%∼12.30% on average. Specifically, at *top-N*=10 in *Video Games* dataset, QUALSE and AQUALSE improves NDCG@10 over the best baseline by 9.9% and 12.97%, respectively.

Figure 5.11(b) shows the HIT@100 performance of our QUALSE and AQUALSE models, and the top-5 baselines in the Yelp dataset when varying the embedding size.



(a) Pet Supplies dataset.

(b) Yelp dataset.

Figure 5.12: Density distribution of item-item similarity scores in train/vad/test sets of *Pet Supplies* and *Yelp* datasets.

We observe that our proposals outperformed all the baselines. Interestingly, while non-adversarial models are more sensitive to the change of the embedding size, our adversarial AQUALSE model is relatively smoother when varying the embedding size. The result makes sense because the adversarial learning reduces the noise effect. Because of the space limitation, we only show detailed results of the *Video Games* and *Yelp* datasets.

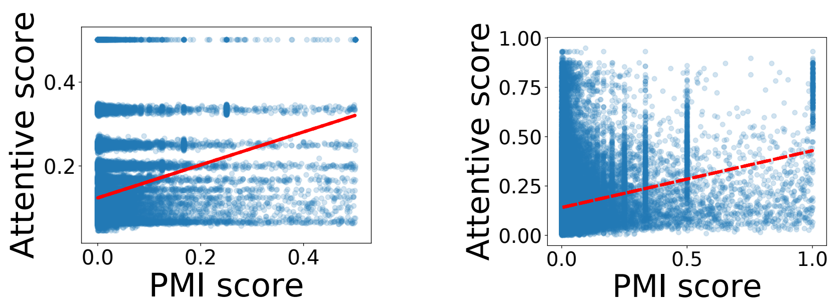
RQ2: Effect of the long-term and short-term encoding components? Using reported results in Table 5.6, we first compare long-term encoding models (i.e. (a)-(d), and *QUALE*, *AQUALE*) with short-term encoding models (i.e. (e)-(h), and *QUASE*, *AQUASE*). In general, long-term encoding models work better than short-term encoding models. For instance, NAIS (i.e. best long-term encoding baseline) improves 8.5% on average on six datasets compared with Caser (i.e. best short-term encoding baseline). Similarly, our long-term encoding *QUALE* model works better than our short-term encoding *QUASE* model, enhancing 9.2% on average over six datasets. To investigate this phenomenon, we plot the density distribution of item-item similarity scores in test sets of two datasets *Pet-Supplies* and *Yelp* in Figure 5.12. We observe higher peaks on long-term item-item relationships in the curves, explaining why long-term encoding models work better than short-term encoding models.

Next, we compare the fused models with models that encode either long-term or short-term users preferences. Table 5.6 shows that models, which consider both user's long-term and short-term preferences, work better than other models, which encode either user's long-term or short-term interests. Both (j) and (k) baselines generally work better than (a)–(h) baselines. Specifically, our QUALSE and AQUALSE models improve 7.9%$\sim$10.0% on average over six datasets compared to the best baseline from (a)–(h). These observations show the effectiveness of modeling both user's long-term and short-term interests. Among models, which consider both user long-term and short-term interests, SASRec performed the worst compared to baselines (i)–(k) and our QUALSE and AQUALSE. This is due to the fact that SASRec models user's long-term and short-term interests implicitly and concurrently by using the Transformer multi-head attention mechanism. But, SLi-Rec, ALSTM+AASVD, and our proposals model the two preferences explicitly and separately, and then combine them later on, increasing flexibility. Note that, although SLi-Rec employed a time-aware attentional LSTM to better model the user's short-term preferences, our ALSTM+AASVD implementation works slightly better than SLi-Rec due to its two distinct properties: (i) the personalized self-attention in AASVD, where each user is parameterized by her own context vector, and (ii) the personalized gating fusion.

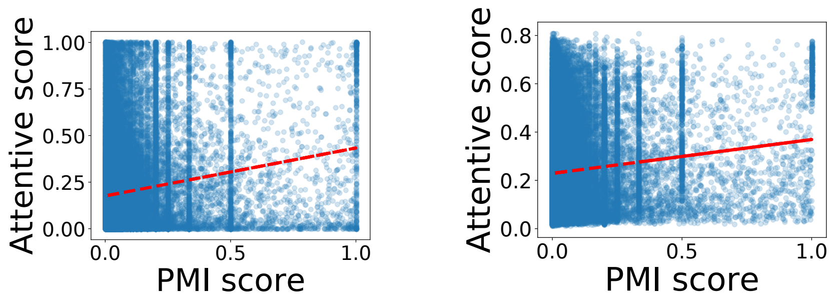
RQ3: Is using Quaternion representation helpful? In Table 5.6, we compare different model pairs: *AASVD* vs. *QUALE*, *ALSTM* vs. *QUASE (LSTM)*, *AGRU* vs. *QUASE (GRU)*, and *ALSTM+AASVD* vs. *QUALSE*. Two methods under the same pair have similar architecture (again, *ALSTM+AASVD* was implemented by us, following our QUALSE architecture to show effectiveness of Quaternion representation). But, the first method of each pair uses real-valued representations and the second method of each pair uses Quaternion representations. Table 5.6 shows that *QUALE* works better than *AASVD*.

In six datasets, on average, *QUALE* improves *Hit@100* by 5.60% and *NDCG@100* by 7.71% compared to *AASVD*. Similarly, we observe the same patterns from the other three model pairs. Moreover, when comparing our long-term encoding *QUALE* and *AQUALE* models with other long-term encoding baselines (a)-(e), our models outperformed the baselines, improving *HIT@100* by 5.16% and 6.55%, and enhancing *NDCG@100* by 7.11% and 9.83%, respectively. Similarly, our short-term encoding *QUASE* and *AQUASE* using LSTM also work better than other short-term encoding baselines (f)-(h), improving *HIT@100* by 4.75% and 8.09%, and enhancing *NDCG@100* by 10.57% and 15.33%, respectively. All of these results confirm the effectiveness of modeling user's interests by using Quaternion representations over Euclidean representations.

Why Quaternion representations help improve the performance? Since attention mechanism is the key success in deep neural networks [113], we analyze how our models assign attention weights compared to their respective real-valued models. We first measure the item-item Pointwise Mutual Information (PMI) scores (i.e. $PMI(j,t) = log\frac{P(j,t)}{P(j)\times P(t)}$) using the training set. The PMI score between two items (j, t) gives us the co-occurrence information between item j and item t, or how likely the target item j will be preferred by the target user when the item t is already in her consumed item list. We perform *softmax* on all item-item PMI scores. Then, we compare with the generated attention scores from our proposed models and ones from their respective real-valued baseline models. Figure 5.13 shows the scatter plots and Pearson correlation comparison using the *Apps for Android* dataset. We see that *QUALE, QUASE* tend to correlate more positively with the PMI scores than their respective real-valued models *AASVD, ALSTM*. In another word, our Quaternion-based models assign higher scores for co-occurred item pairs. We reason coming from two aspects of Quaternion representations. First, Hamilton product in Quaternion space encourages strong inter-latent interactions across Quaternion components. Second, since our proposed self-attention mechanism produces



(a) QUALE (left) vs AASVD (right).



(b) QUASE (left) vs ALSTM (right).

Figure 5.13: Comparison of attention scores between (QUALE vs AASVD) and (QUASE vs ALSTM) in the Apps for Android dataset. Pearson correlation ρ between attention scores and PMI scores are: $\rho_{QUALE} = 0.232 > \rho_{AASVD} = 0.216$, and $\rho_{QUASE} = 0.148 > \rho_{ALSTM} = 0.1$.

scores in Quaternion space, the output attention scores have four values *w.r.t* four Quaternion components. This can be thought as similar to the multi-head attention mechanism [113] (but not exactly same because of the weight shared in Quaternion transformation), where the proposed attention mechanism learns to attend different aspects from the four Quaternion components. All of these explain why we got better results compared to the respective real-valued models.

RQ4: Effect of the personalized gated fusion and the QABPR loss? Table 5.6 shows that in real-valued representations, *ALSTM+ASVD* works better than *AASVD* and *ALSTM* in all six datasets. Similarly, in Quaternion representations, the fused *QUALSE* model generally works better than its two degenerated *QUALE* and *QUASE* models. In the six datasets, both *QUALSE* and *AQUALSE* perform better than their degenerated (adversarial) versions, improving 2% on average *w.r.t* both *HIT@100* and *NDCG@100*. The results confirm the effectiveness of fusing long-term and short-term user preferences in both of *QUALSE* and *AQUALSE*.

We further compare our gating fusion with a weight fixing method, where we vary a contribution score $c \in [0, 1]$ for the user's short-term preference encoding part and $1 - c$ for the long-term part. We see that the gating fusion improves 4.82% on average over six datasets compared to the weight fixing method, again confirming the effectiveness of our personalized gating fusion method.

Is Quaternion Adversarial training on BPR loss helpful? We compare our proposed models training with *BPR* loss (i.e. *QUALE, QUASE (LSTM),* and *QUALSE* models) and our proposed models training with *QABPR* loss (i.e. *AQUALE*, *AQUASE (LSTM)*, and *AQUALSE*). First, we observe that *AQUASE* boosted *QUASE* performance by a large margin: improving *HIT@100* by 3.2% and *NDCG@100* by 4.29% on average in the six datasets. *AQUALE* and *AQUALSE* also improve *QUALE* and *QUALSE* by 1.92% and 1.91% on average of both *HIT@100* and *NDCG@100* over six datasets, respectively. These results show the effectiveness of the adversarial attack on Quaternion representations with our *QABPR* loss.

6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, we observe two groups of users – consumers and product owners in online systems such as e-commerce systems (e.g., Amazon, eBay) and streaming service systems (e.g., Netflix, Youtube, Spotify). We have developed algorithms and techniques to model heterogeneous user behaviors. In particular, this dissertation has made three unique contributions as follows:

First, we characterize the product deliver behavior of product owners. We proposed a clustering approach to group rewards by their latent difficulty levels. Based on the analysis and study, we extracted various features toward building predictive models for a reward delivery status (i.e., on-time or late) and delivery duration.

Second, we study the heterogeneous consumer behaviors via their global interests. In this direction, we proposed to exploit different co-occurrence information: co-disliked item-item co-occurrences, co-liked item-item co-occurrences, and consumer-consumer co-occurrences, which were extracted from the consumer-item interaction matrix. We proposed a joint model combining WMF, co-liked embedding, co-disliked embedding and consumer embedding, following the recent success of word embedding techniques. Through comprehensive experiments, we successfully demonstrated that our model outperformed all baselines. We also analyzed how our model worked on different types of consumers in terms of their interaction activity levels. We observed that our model significantly improved the state-of-the-art compared models for the *cold-start users* group. Moreover, due to the fact that matrix factorization based methods have intrinsic linear nature, which is limited in modeling complex consumer-item relationships, we further proposed three novel recommendation approaches based on Mahalanobis distance and performed experiments on the automatic playlist continuation problem in the streaming platforms. Our *MDR* model used Mahalanobis distance to account for both consumers' preferences and playlists' themes over songs. Our *MASS* model measured attentive similarities between a candidate song and member songs in a target playlist through our proposed memory metric-based attention mechanism. Our *MASR* model combined the capabilities of *MDR* and *MASR*. We also adopted and customized *Adversarial Personalized Ranking* (APR) loss with proposed flexible noise magnitude to further enhance the robustness of our three models. Through extensive experiments against several state-of-the-art baselines, we showed that our proposals were not only effective but also efficient compared to the baseline models.

Third, we explore the consumers' behavior via their long-term and short-term preference footprints. In this direction, we have considered two independent signed distance models for measuring consumer-item and item-item similarities, respectively, via deep neural networks. The proposed SDP learns a non-linear metric in a consumer-item latent space, while SDM learns a personalized item-item distance with soft attention. Then the two networks are combined for a compounded signed distance approximator called SDMR. Extensive experiments have been performed on six real-world datasets in general recommendation and shopping basket-based recommendation task. We presented that our proposed SDMR outperformed ten baselines in all two recommendation tasks. Furthermore, as Quaternion space has shown its superior benefits/performance over the traditional Euclidean space in NLP and computer vision, we put a step further and fully utilized Quaternion space and proposed three novel Quaternion-based recommendation models: (i) a *QUALE* model learned the consumer's long-term intents, (ii) a *QUASE* model learned the consumer's short-term interests, and (iii) a *QUALSE* model fused *QUALE* and *QUASE* to learn both consumer's long-term and short-term preferences. We also proposed a Quaternion-based Adversarial attack on Bayesian Personalized Ranking (QABPR) loss to improve the robustness of our proposals. Through extensive experiments on six real-world datasets, we showed that our proposed models achieved the best results compared to the baseline models.

6.2 Future Work

In this section, we depict some extension of our current work as follows:

Side Information Utilization with Metric Learning: In this dissertation, we mainly focused on modeling user behaviors using only user-item interaction data. In fact, there exists additional information that we can utilize to improve the performance of our proposed approaches such as user's reviews, user's rating scores, product's reviews, product's images, and product's description, *etc*. In short, this auxiliary information can be grouped into three different formats: texts, images, and video. Even though there existed studies that took into account item description [53], item visual features [55, 56], and user reviews [57] for recommendation systems, they still adopted the inner product for measuring user-item similarity scores. Thus, applying metric learning on the learned feature representations from the side information could further improve the performance of our proposed models. Moreover, with the great success of recent language models such as

ELMo [141], BERT [142], XLNet [143], GPT [144], modeling consumer behaviors with additional textual information using the state-of-the-art language models becomes an even more exciting research direction.

Explainability. Diversity, Freshness and Fairness Awareness: The usual approach in modeling consumer behaviors in this dissertation was to focus on the relevance of an item to a target consumer, that is, to what degree the recommending item matched the consumer taste. However, as depicted in Chapter 5 that the consumer's interests are highly dynamic, and the consumers are likely to experience with diverse and fresh items [145, 146, 147]. Following this observation, we argue here that the consumer's behaviors are driven not only by relevance but also by diversity and freshness needs. Also, explaining why a consumer likes an item is as important as the recommendation accuracy. Thus, establishing explainable and transparent recommenders is necessary to improve the models' persuasiveness and trustworthiness. Moreover, designing fairness-aware recommender systems is another interesting line of research such that recommending products for customers do not discriminate against individuals or groups. All into consideration, integrating our proposals with explainability, diversity, freshness and fairness awareness will be even a more exciting extension for our work presented here.

References

[1] Paul Belleflamme, Thomas Lambert, and Armin Schwienbacher. Crowdfunding: Tapping the right crowd. *JBV*, 2012. 10

[2] Elizabeth M. Gerber and Julie Hui. Crowdfunding: Motivations and deterrents for participation. *TOCHI*, 2013. 10

[3] Venkat Kuppuswamy and Barry L Bayus. Crowdfunding creative ideas: The dynamics of project backers in kickstarter. *UNC Kenan-Flagler Research Paper*, 2015. 10, 15

[4] Ethan Mollick. The dynamics of crowdfunding: An exploratory study. *JBV*, 2014. 10

[5] Elizabeth M Gerber, Julie S Hui, and Pei-Yi Kuo. Crowdfunding: Why people are motivated to post and fund projects on crowdfunding platforms. In *CSCW*, 2012. 10, 15

[6] Thanh Tran, Madhavi R Dontham, Jinwook Chung, and Kyumin Lee. How to succeed in crowdfunding: a long-term study in kickstarter. *CoRR*, 2016. 10, 11, 15

[7] Anbang Xu, Xiao Yang, Huaming Rao, Wai-Tat Fu, Shih-Wen Huang, and Brian P Bailey. Show me the money!: an analysis of project updates during crowdfunding campaigns. In *CHI*, 2014. 10, 15

[8] Dieter W Joenssen and Thomas Müllerleile. Limitless crowdfunding? the effect of scarcity management. In *Crowdfunding in Europe*, pages 193–199, 2016. 10

[9] Julie S Hui, Michael D Greenberg, and Elizabeth M Gerber. Understanding the role of community in crowdfunding work. In *CSCW*, 2014. 10, 15

[10] Chun-Ta Lu, Hong-Han Shuai, and Philip S Yu. Identifying your customers in social networks. In *CIKM*, pages 391–400, 2014. 10, 15

[11] Chun-Ta Lu, Sihong Xie, Xiangnan Kong, and Philip S Yu. Inferring the impacts of social media on crowdfunding. In *WSDM*, 2014. 10, 15

[12] Dieter William Joenssen, Anne Michaelis, and Thomas Müllerleile. A link to new product preannouncement: Success factors in crowdfunding. *SSRN*, 2014. 10

[13] Vincent Etter, Matthias Grossglauser, and Patrick Thiran. Launch hard or go home!: predicting the success of kickstarter campaigns. In *COSN*, 2013. 11, 15

[14] Michael D Greenberg, Bryan Pardo, Karthic Hariharan, and Elizabeth Gerber. Crowdfunding support tools: predicting success & failure. In *CHI*, 2013. 11, 15

[15] Jinwook Chung and Kyumin Lee. A long-term study of a crowdfunding platform: Predicting project success and fundraising amount. In *HT*, 2015. 11, 15

[16] Yan Li, Vineeth Rakesh, and Chandan K Reddy. Project success prediction in crowdfunding environments. In *WSDM*, 2016. 11, 15

[17] Jacob Solomon, Wenjuan Ma, and Rick Wash. Don't wait!: How timing affects coordination of crowdfunding donations. In *CSCW*, 2015. 11

[18] Tanushree Mitra and Eric Gilbert. The language that gets people to give: Phrases that predict success on kickstarter. In *CSCW*, 2014. 11

[19] Jisun An, Daniele Quercia, and Jon Crowcroft. Recommending investors for crowdfunding projects. In *WWW*, 2014. 11, 15

[20] Vineeth Rakesh, Jaegul Choo, and Chandan K Reddy. Project recommendation using heterogeneous traits in crowdfunding. In *ICWSM*, 2015. 11

[21] Vineeth Rakesh, Wang-Chien Lee, and Chandan K Reddy. Probabilistic group recommendation model for crowdfunding domains. In *WSDM*, 2016. 11, 15

[22] Yongsung Kim, Aaron Shaw, Haoqi Zhang, and Elizabeth Gerber. Understanding trust amid delays in crowdfunding. In *CSCW*, 2017. 11, 27, 29, 32

[23] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008. 11, 34, 38, 39, 50, 86, 89

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009. 11, 34, 86

[25] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. In *SIGIR*, pages 325–334, 2016. 11

[26] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017. 12, 46, 49, 76, 77, 78, 86, 89, 90, 91, 94, 106, 108, 131

[27] Lucas Vinh Tran, Tuan-Anh Nguyen Pham, Yi Tay, Yiding Liu, Gao Cong, and Xiaoli Li. Interact and decide: Medley of sub-attention networks for effective group recommendation. In *SIGIR*, pages 255–264, 2019. 12

[28] Thanh Tran, Renee Sweeney, and Kyumin Lee. Adversarial mahalanobis distance-based attentive song recommender for automatic playlist continuation. In *SIGIR*, pages 245–254, 2019. 12

[29] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. In *SIGIR*, 2018. 12, 76, 107

[30] Chen Ma, Yingxue Zhang, Qinglong Wang, and Xue Liu. Point-of-interest recommendation: Exploiting self-attentive autoencoders with neighbor-aware influence. In *CIKM*, pages 697–706, 2018. 12

[31] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *WWW*, pages 689–698, 2018. 12, 89, 104, 133

[32] Chen Ma, Peng Kang, Bin Wu, Qinglong Wang, and Xue Liu. Gated attentive-autoencoder for content-aware recommendation. In *WSDM*, pages 519–527, 2019. 12

[33] Xin Xin, Xiangnan He, Yongfeng Zhang, Yongdong Zhang, and Joemon Jose. Relational collaborative filtering: Modeling multiple item relations for recommendation. In *SIGIR*, pages 125–134, 2019. 12, 133

[34] Thanh Tran, Xinyue Liu, Kyumin Lee, and Xiangnan Kong. Signed distance-based deep memory recommender. In *WWW*, pages 1841–1852, 2019. 12, 77

[35] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*, pages 426–434, 2008. 12, 86, 88, 108, 121, 131

[36] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *KDD*, pages 659–667, 2013. 12, 70, 76, 86, 133

[37] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and

Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE TKDE*, 30(12):2354–2366, 2018. 12, 132

[38] Chen Ma, Peng Kang, and Xue Liu. Hierarchical gating networks for sequential recommendation. In *KDD*, pages 825–833, 2019. 12

[39] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new poi recommendation. In *IJCAI*, volume 15, pages 2069–2075, 2015. 12, 61, 76, 91, 97, 107

[40] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pages 811–820, 2010. 12, 86, 91, 108, 132

[41] Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation. In *RecSys*, pages 161–169, 2017. 12, 61, 77, 86, 98, 107

[42] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*, pages 565–573, 2018. 12, 77, 86, 108, 132

[43] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv*, 2015. 12, 86, 91, 108, 132

[44] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *WSDM*, pages 495–503, 2017. 12, 124

[45] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *CIKM*, pages 1419–1428, 2017. 12

[46] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. Context-aware sequential recommendation. In *ICDM*, pages 1053–1058, 2016. 12

[47] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *ICDM*, pages 197–206, 2018. 12, 86, 130, 132

[48] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *AAAI*, volume 33, pages 5941–5948, 2019. 12, 117, 124, 133

[49] Zeping Yu, Jianxun Lian, Ahmad Mahmoody, Gongshen Liu, and Xing Xie. Adaptive user modeling with long and short-term preferences for personalized recommendation. In *IJCAI*, pages 4213–4219, 2019. 12, 117, 132

[50] Yichao Lu, Ruihai Dong, and Barry Smyth. Convolutional matrix factorization for recommendation explanation. In *IUI*, page 34, 2018. 12

[51] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. *arXiv*, 2018. 12

[52] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. Interpretable convolutional neural networks with dual local and global attention for review rating prediction. In *RecSys*, pages 297–305, 2017. 12

[53] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *RecSys*, pages 233–240, 2016. 12, 89, 143

[54] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NeurIPS*, pages 2643–2651, 2013. 12, 89

[55] Qiang Liu, Shu Wu, and Liang Wang. Deepstyle: Learning user preferences for visual recommendation. In *SIGIR*, pages 841–844, 2017. 12, 89, 143

[56] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *SIGIR*, pages 335–344, 2017. 12, 89, 143

[57] Yichao Lu, Ruihai Dong, and Barry Smyth. Coevolutionary recommendation model: Mutual learning between ratings and reviews. In *WWW*, pages 773–782, 2018. 12, 89, 143

[58] Tim Althoff and Jure Leskovec. Donor retention in online crowdfunding communities: A case study of donorschoose. org. In *WWW*, 2015. 15

[59] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014. 20

[60] Wikipedia. Smog grade. `https://en.wikipedia.org/wiki/SMOG`, 1969. 24

[61] James W Pennebaker, Roger J Booth, and Martha E Francis. Linguistic inquiry and word count: Liwc [computer software]. *Austin, TX: liwc. net*, 2007. 25

[62] Kyumin Lee, Jalal Mahmud, Jilin Chen, Michelle Zhou, and Jeffrey Nichols. Who will retweet this?: Automatically identifying and engaging strangers on twitter to spread information. In *IUI*, 2014. 25

[63] Kyumin Lee, Prithivi Tamilarasan, and James Caverlee. Crowdturfers, campaigns, and social media: Tracking and revealing crowdsourced manipulation of social media. In *ICWSM*, 2013. 25

[64] Miron B Kursa, Witold R Rudnicki, et al. Feature selection with the boruta package, 2010. 26

[65] Michael H Kutner, Chris Nachtsheim, and John Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004. 30

[66] Seyed Reza Shahamiri and Siti Salwah Binti Salim. Real-time frequency-based noise-robust automatic speech recognition using multi-nets artificial neural networks: A multi-views multi-learners approach. *Neurocomputing*, 2014. 32

[67] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009. 34

[68] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994. 34, 49, 104

[69] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *KDD*, pages 19–28, 2009. 34

[70] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011. 34

[71] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. Dynamic matrix factorization with priors on unknown values. In *KDD*, pages 189–198, 2015. 34

[72] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *RecSys*, pages 71–78, 2010. 34, 46

[73] Maksims Volkovs and Guang Wei Yu. Effective latent models for binary feedback in recommender systems. In *SIGIR*, pages 313–322, 2015. 34, 46

[74] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*, pages 549–558, 2016. 35, 46, 86, 89, 90, 131

[75] Marcel Blattner, Yi-Cheng Zhang, and Sergei Maslov. Exploring an opinion network for taste prediction: An empirical study. *Physica A: Statistical Mechanics and its Applications*, pages 753–758, 2007. 37

[76] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *RecSys*, pages 59–66, 2016. 39, 40, 49, 50

[77] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013. 39, 50

[78] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NeurIPS*, pages 2177–2185, 2014. 39

[79] Elie Guàrdia-Sebaoun, Vincent Guigue, and Patrick Gallinari. Latent trajectory modeling: A light and efficient way to introduce time in recommender systems. In *RecSys*, pages 281–284, 2015. 40

[80] Oren Barkan and Noam Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *MLSP Workshop*, pages 1–6, 2016. 40, 50

[81] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009. 43

[82] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Parallel matrix fac-

torization for recommender systems. *Knowledge and Information Systems*, pages 793–819, 2014. 43, 46

[83] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348, 2008. 43

[84] Harald Steck. Training and testing of recommender systems on data missing not at random. In *KDD*, pages 713–722, 2010. 46

[85] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*, pages 785–788, 2013. 46

[86] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *ICDM*, pages 502–511, 2008. 46

[87] Bing Liu, Wee Sun Lee, Philip S Yu, and Xiaoli Li. Partially supervised classification of text documents. In *ICML*, pages 387–394, 2002. 46

[88] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016. 49, 76, 104, 130, 131

[89] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, 2004. 50

[90] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001. 50, 77, 89, 106

[91] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. Modeling user exposure in recommendation. In *WWW*, pages 951–961, 2016. 51

[92] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *WWW*, pages 193–201, 2017. 60, 61, 76, 97, 98, 106

[93] Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *KDD*, pages 931–939, 2012. 60, 98

[94] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *KDD*, pages 714–722, 2012. 61

[95] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *NeurIPS*, pages 1473–1480, 2006. 61

[96] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NeurIPS*, pages 521–528, 2003. 61

[97] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. Adversarial personalized ranking for recommendation. In *SIGIR*, pages 355–364, 2018. 61, 73, 74, 78, 80, 82, 128, 129

[98] Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009. 64

[99] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NeurIPS*, pages 2222–2230, 2012. 68, 96

[100] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *ICML*, pages 2397–2406, 2016. 68, 95

[101] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *ICCV*, pages 2425–2433, 2015. 68

[102] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, pages 315–323, 2011. 69

[103] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 11th IEEE International Conference on Data Mining*, pages 497–506, 2011. 70, 77, 89, 106

[104] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *NeurIPS*, pages 2440–2448, 2015. 70, 95, 97, 100

[105] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*, pages 1378–1387, 2016. 70

[106] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv*, 2016. 70

[107] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv*, 2016. 74, 128, 129

[108] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 30music listening and playlists dataset., 2015. 75

[109] B. McFee and G. R. G. Lanckriet. Hypergraph models of playlist dialects. In *ISMIR*, 2012. 75

[110] Thanh Tran, Kyumin Lee, Yiming Liao, and Dongwon Lee. Regularizing matrix factorization with user and item embeddings for recommendation. In *CIKM*, pages 687–696, 2018. 76, 104, 133

[111] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009. 76, 106, 133

[112] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv*, 2015. 83, 98

[113] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 6000–6010, 2017. 83, 98, 113, 138, 139

[114] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997. 86

[115] Xinyue Liu, Chara Aggarwal, Yu-Feng Li, Xiaugnan Kong, Xinyuan Sun, and Saket Sathe. Kernelized matrix factorization for collaborative filtering. In *SDM*, pages 378–386, 2016. 89

[116] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*, pages 153–162, 2016. 89

[117] Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *CIKM*, pages 811–820, 2015. 89

[118] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW*, pages 111–112, 2015. 89

[119] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*, pages 130–137, 2017. 91

[120] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *Proceeding of the 26th International Joint Conference on Artificial Intelligence*, pages 3203–3209, 2017. 94, 108

[121] Fei Liu and Julien Perez. Gated end-to-end memory networks. In *EACL*, volume 1, pages 1–10, 2017. 95, 100

[122] Hanwang Zhang, Yang Yang, Huanbo Luan, Shuicheng Yang, and Tat-Seng Chua. Start from scratch: Towards automatically identifying, modeling, and naming visual attributes. In *MM*, pages 187–196, 2014. 96

[123] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NeurIPS*, pages 2321–2329, 2014. 98

[124] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv*, 2017. 98

[125] Heeyoul Choi, Kyunghyun Cho, and Yoshua Bengio. Fine-grained attention mechanism for neural machine translation. *Neurocomputing*, 284:171–176, 2018. 98

[126] Paul Hongsuck Seo, Zhe Lin, Scott Cohen, Xiaohui Shen, and Bohyung Han. Hierarchical attention networks. *arXiv*, 2016. 98

[127] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014. 98, 115

[128] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, pages 2048–2057, 2015. 98, 115

[129] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. In *WWW*, pages 729–739, 2018. 99, 109

[130] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 103, 109, 133

[131] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *RecSys*, pages 17–24, 2007. 104

[132] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for nextbasket recommendation. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 403–412, 2015. 108

[133] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 191–200, 2016. 108

[134] Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linarès, Chiheb Trabelsi, Renato De Mori, and Yoshua Bengio. Quaternion recurrent neural networks. In *ICLR*, 2019. 120, 124

[135] Chase J Gaudet and Anthony S Maida. Deep quaternion networks. In *IJCNN*, pages 1–8, 2018. 120

[136] Victor Zhong, Caiming Xiong, and Richard Socher. Global-locally self-attentive encoder for dialogue state tracking. In *ACL*, pages 1458–1467, 2018. 124

[137] Shuai Zhang, Lina Yao, Lucas Vinh Tran, Aston Zhang, and Yi Tay. Quaternion collaborative filtering for recommendation. In *IJCAI*, pages 4313–4319, 2019. 124, 131

[138] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *KDD*, pages 1933–1942, 2017. 124

[139] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 127

[140] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019. 133

[141] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv*, 2018. 144

[142] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv*, 2018. 144

[143] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5753–5763, 2019. 144

[144] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv*, 2020. 144

[145] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005. 144

[146] Rubi Boim, Tova Milo, and Slava Novgorodov. Diversification and refinement in collaborative filtering recommender. In *CIKM*, pages 739–744, 2011. 144

[147] Idan Szpektor, Yoelle Maarek, and Dan Pelleg. When relevance is not enough: promoting diversity and freshness in personalized question recommendation. In *WWW*, pages 1249–1260, 2013. 144