Learning Implicit Recommenders from Massive Unobserved Feedback

Fajie Yuan

School of Computing Science

College of Science and Engineering

University of Glasgow

This dissertation is submitted for the degree of

Doctor of Philosophy (PhD)

14^{th} Sep 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University.

This dissertation is the result of my own work, under the supervision of Professor Joemon M. Jose and Dr. Simon Rogers.

Fajie Yuan
Sep, 2018

Acknowledgements

First of all, I would like to express my sincere gratitude to my doctoral supervisor Prof. Joemon M. Jose. Without your careful instruction, I would never achieve so much success and finish the degree. Joemon is not only a nice and easy-going professor, but also one of the best friends in my life. Thanks for your support and encouragement. Thanks for believing in me and teaching me. With your supervision, I really enjoy my research and my Ph.D life.

Second, I would like to show my great respect to my second supervisor Dr. Simon Rogers. Thanks for agreeing to be my second supervisor. Simon is a cool guy and a helpful friend. Thanks for giving me so much good advice in my annual review.

Third, I want to thank Dr. Guibing Guo, Dr. Weinan Zhang and Dr. Xiangnan He, who I have worked with for many years. They are the most outstanding young researchers in my field. I have learned a lot from him during collaboration and got many help from them. Without their help, I will not have so many good work accepted by prestigious conferences.

Fourth, I would like to thank my internship advisor Dr. Alexandros Karatzoglou and Ioannis Arapakis in Telefonica Research. Alexandros and Ioannis are great researchers in industry and gave me so much invaluable advice in guiding me towards doing useful and successful research.

Fifth, I would like to thank my two examiners Prof. Pablo Castells and Dr. Ke Yuan for their expert advice to further improve my thesis.

Sixth, I would like to thank my university colleagues: Stewart, Rami, Philip, James, Jesus, Fatma, Jarana, David Maxwell, Jorge Paule, Stuart Mackie, Yashar Moshfeghi, Nujud and Xin Xin. Particularly, thanks Stewart for many help on my research.

Seventh, I would like to thank my best Chinese friends in Glasgow: Ya-

meng, Yaqi, Zhilong, Pengcheng, Xinxin, Yingcheng, Yuting, Xiaocai, Haitao and Yushu. Thanks guys for giving me so much happiness!

Last but not least, I would like to acknowledge the most important people in my life — my parents and my wife. I want to give my best thanks to them for their love and unconditional supports during these years!

Abstract

Recommender systems have become an attractive research topic during the last two decades, and are utilized in a broad range of areas, including movies, music, images, advertisements, natural language, search engines, and social websites, etc. Early recommendation research largely focused on explicit feedback such as user ratings. However, in practice most observed user feedback is not explicit but implicit, such as clicks, purchases and dwell time, where users' explicit preference on items is not provided. Moreover, implicit feedback is usually tracked automatically and, thus, it is more prevalent, inexpensive and scalable.

In this thesis we investigate implicit feedback techniques for real-world recommender systems. However, learning a recommender system from implicit feedback is very challenging, primarily due to the lack of negative feedback. While a common strategy is to treat the unobserved feedback (i.e., missing data) as a source of negative signal, the technical difficulties cannot be overlooked: (1) the ratio of positive to negative feedback in practice is highly imbalanced, and (2) learning through all unobserved feedback (which easily scales to billion level or higher) is computationally expensive.

To effectively and efficiently learn recommender models from implicit feedback, two types of methods are presented, that is, negative sampling based stochastic gradient descent (NS-SGD) and whole sample based batch gradient descent (WS-BGD). Regarding the NS-SGD method, how to effectively sample informative negative examples to improve recommendation algorithms is investigated. More specifically, three learning models called Lambda Factorization Machines (lambdaFM), Boosting Factorization Machines (BoostFM) and Geographical Bayesian Personalized Ranking (GeoBPR) are described. While regarding the WS-BGD method, how to efficiently use all unobserved implicit feedback data rather than resorting to negative sampling is studied. A fast BGD learning

algorithm is proposed, which can be applied to both basic collaborative filtering and content/context-aware recommendation settings.

The last research work is on the session-based item recommendation, which is also an implicit feedback scenario. However, different from above four works based on shallow embedding models, we apply deep learning based sequence-to-sequence model to directly generate the probability distribution of next item. The proposed generative model can be applied to various sequential recommendation scenarios.

To support the main arguments, extensive experiments are carried out based on real-world recommendation datasets. The proposed recommendation algorithms have achieved significant improvements in contrast with strong benchmark models. Moreover, these models can also serve as generic solutions and solid baselines for future implicit recommendation problems.

Table of Contents

List of Figures

List of Tables

Part I
Introduction and Background

The thesis focuses on the item recommendation task from implicit feedback. The main research question is how to effectively use the large-scale unobserved feedback. This part includes Chapter 1 for introduction and Chapter 2 for implicit feedback model reviews. In Chapter 1, we have surveyed the background information for implicit feedback based item recommendation, and then provided thesis motivations, statements, challenges and contributions. In Chapter 2, we investigated the most widely used prediction models, objective functions and evaluation metrics for item recommendation from implicit feedback.

Chapter 1

Introduction

1.1 Background on Recommendation

In the last two decades, the emerging growth of the Internet has brought a huge amount of digital information, including but not limited to, texts, images, audios/videos and electronic products. Meanwhile, a variety of social networks (e.g., Facebook[1] and Weichat[2]), electronic business platforms (e.g., Amazon[3]), and multimedia sharing websites (e.g., Youtube[4]) have become more and more popular. The volume of online information is growing at an unbelievably fast rate, since a large number of visitors in these application platforms can frequently create, upload and share new information (Shi, 2013). With such huge amount of information, it is difficult for users to find useful items and make effective decisions, which is referred to as information overload[5]. To deal with this issue, two important information processing technologies have been developed (Belkin and Croft, 1992): information retrieval and recommendation.

In an information retrieval (IR) system, users explicitly present their information needs by sending a query to a search engine (e.g., Google[6]) (Costa and Roda, 2011; Belkin and Croft, 1992). The system processes the request by searching existing resources and returns a list of matched items according to a certain retrieval algorithm. On the contrary, in a recommendation platform, the system (e.g., Amazon) typically automatically generates recommendations based on

[1] https://www.facebook.com/
[2] https://web.wechat.com/
[3] https://www.amazon.co.uk//
[4] https://www.youtube.com
[5] https://en.wikipedia.org/wiki/Information_overload
[6] https://www.google.co.uk/

users' profiles rather than an explicit query. Both ways play an important role in obtaining knowledge from the Internet. The main difference lies in that the needs of a user in an IR system are expressed explicitly, whereas they are typically unspecified in a recommender system (RS), and as a result, the user is more likely to passively accept recommended information (Shi, 2013). In addition, compared to search engines, recommender systems help users discover new unexpected items, or support serendipitous (Kawamae, 2010) discoveries, which they may not initially realize when formulating the query (Tintarev et al., 2013; Shi, 2013). A good IR system typically needs to have accuracy guarantee in matched items, which, however, is not enough for a RS system. Recommendation accuracy, personalization, and context are all important factors for evaluating a RS system. In this thesis, our main research focus is to study and develop new advancement in recommender systems, and address critical research challenges in this domain.

A recommender system is formally defined as "*a subclass of information filtering system that seeks to predict the 'rating' or 'preference' a user would give to an item*" (Ricci et al., 2011). Recommender systems are important in assisting users with their choices. A good recommender system is supposed to suggest the right items to the right person in the right context. In recent years recommender systems have been widely used in a variety of domains (Sheth et al., 2010), such as recommending movies we might like (Katarya and Verma, 2017), places we might visit (Yuan et al., 2016d), things we might buy (Yuan et al., 2018a), friends we may know (Yuan et al., 2016a) and queries we may type (Baeza-Yates et al., 2004).

There are two types of recommender systems according to the way to generate recommendations: collaborative filtering (Koren and Bell, 2015) and content/context based recommendation (Rendle, 2012). Collaborative filtering (aka neighborhood-based) approaches build the recommendation model based on users' past histories, or decisions made by other similar users (i.e., neighbors). These approaches are typically based on a very simple recommendation algorithm, which does not rely on the features or contents of users and items. For example, the recommender system in Amazon can generate a list of products by leveraging only the collective buying or rating behaviors of various users. The well-known recommendation principle of collaborative filtering is that similar users tend to be

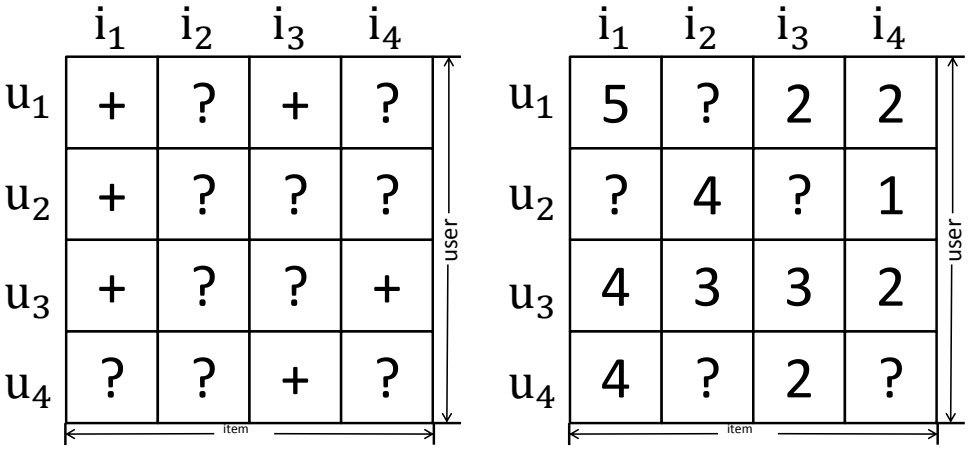
interested in similar items (Aggarwa, 2016). However, in practice, recommender systems can be more complex and have to consider more auxiliary characteristics for users and products. This leads to the content/context based recommendation approaches, which leverage additional attributes such as the ratings of users or descriptions of items. For example, the Pandora[1] recommender system suggests music by leveraging the properties of a song or artist (around 400 attributes[2]), and play music that has similar properties to the user. The basic idea is that users' preference can be modeled through attributes of the items that they liked in the past. We (Yuan et al., 2016b,d, 2017, 2018b) have proven that leveraging additional content (or context) information of users and items can largely improve the performance of recommender systems. Therefore, in this thesis we study recommender systems by considering both collaborative and content/context information.

1.2 Implicit Feedback Recommendation

Recommender systems can also be classified by the type of input data. There is one line of research that focuses on explicit feedback, the data of which explicitly indicates the degree of a user's preference on an item. The most common explicit feedback data is users' ratings, such as a 1-to-5 score or a thumbs-up/down mark. The recommendation task that is based on explicit feedback data is often referred to as the rating prediction problem, where items with higher rating scores are recommended to the user by priority. However, users in practice do not always deliberately choose to rate items. Hence, explicit feedback is not always available (Rendle et al., 2009b; Rendle and Freudenthaler, 2014).

By contrast, implicit feedback is much easier to collect since it can be tracked automatically. Examples of implicit feedback include users' clicks on webpages, purchases of products, dwell time, and video views, where users' explicit preference on items is not provided. Clearly, implicit feedback data is more prevalent, inexpensive and scalable than explicit rating data. For example, even though there is no rating data provided, Netflix[3] at least knows whether the user has

[1] https://uk.pandora.net/en
[2] https://en.wikipedia.org/wiki/Recommender_system
[3] https://www.netflix.com/gb/



(a) Implicit matrix (b) Explicit matrix

Figure 1.1: Sparse matrices of implicit (a) and explicit (b) data. u and i denotes user and item, respectively. "+" and "?" denote positive (e.g., a click) and unobserved feedback (i.e., no click), respectively. The numerical values in (b) represent explicit rating scores that users assigned to items, while on (a), users' explicit feedback (i.e., ratings) is not observed.

watched the movie or not with observed feedback. Recommendation based on implicit feedback is usually formulated as the item recommendation task, which is essentially a ranking task (Rendle et al., 2009b; Yuan et al., 2016b). Moreover, compared with the rating prediction task, implicit feedback based item recommendation task is more close to real-world recommendation scenarios since an observed positive feedback can be reasonably presumed as the user's interests on the item, whereas the difference of ratings does not directly indicate whether the user wants to buy or click the item. One reason is probably because most ratings are provided after the fact that the user has already checked out the item, while the recommendation process is often finished before the user's check-out behavior. Figure 1.1 shows the difference between implicit and explicit feedback input data.

The main focus of this thesis is on item recommendations from implicit feedback. Based on the above discussion, several characters for implicit feedback data can be identified: (1) only positive user feedback (e.g., an observed click or purchase) is available, while negative and unknown (or missing) feedback are

mixed together (see Figure 1.1(a)); (2) the ratio of positive to unobserved negative feedback is highly unbalanced. Most users only interact with a small portion of items, which results in a very sparse user-item matrix (the sparsity in many real-world recommender systems is higher than 95% (Grčar et al., 2005)); (3) the amount of unobserved feedback has an extremely large scale. For example, assuming that there is a recommender system consisting of 1 million users and items, the total implicit feedback contains 1 trillion user-item pairs. According to (2), most (i.e., 95%) of the 1 trillion implicit feedback is unobserved. Thus, learning from implicit feedback is very challenging, and differs from standard machine learning tasks with both positive and negative data. Specifically, the main challenges that we focus on in this thesis lie in the following aspects for the implicit recommendation problem:

- How to learn recommender models from one-class (i.e., positive-only) data?
- How to distinguish negative examples from unobserved feedback?
- How to learn recommender models from highly imbalanced data?
- How to efficiently learn recommender models from large-scale unobserved data?

The thesis targets at above-mentioned four challenges and propose a series of solutions to address them.

1.3 Thesis Statement

The statement of this thesis is that unobserved examples (i.e., user-item pairs) from implicit feedback data have an important impact on the performance of recommender systems. First, understanding what kinds of unobserved examples are more informative will help build more effective recommendation algorithms. Second, the way how to leverage unobserved examples (sampling vs. non-sampling) directly influences the performance of a recommendation algorithm. In other words, the convergence, training time and prediction accuracy of a recommendation model are largely determined by the sampling distribution and size of

unobserved samples. The final important statement is that leveraging all unobserved examples without sampling shows state-of-the-art recommendation accuracy, and empirically outperforms many sampling based recommendation algorithms. Overall, the statements set forth by the thesis are as follows:

- **Statement (1)**: Unobserved items that are ranked higher (by recommendation algorithms) in the recommendation list are more informative than those lower ranked items. According to this, we may infer that unobserved items with higher popularity are more informative than those with lower popularity. The reason is that items with higher popularity are more likely to be ranked at the top position in the ranking list as they have more chances to be positive items and positive items are supposed to be ranked higher.
- **Statement (2)**: According to (1), sampling unobserved items with higher popularity or ranks will largely improve the recommendation accuracy on various top-N ranking measures.
- **Statement (3)**: Sampling more than one unobserved item for each positive item helps improve the recommendation accuracy in various recommendation measures. However, it will also increase the computational complexity as more unobserved items need more stochastic gradient descent (SGD) updates.
- **Statement (4)**: Although sampling methods are widely used in the implicit feedback setting, they are suboptimal as it cannot converge to the same loss as all unobserved items in the list. Training with all unobserved examples empirically shows better performance than sampling based methods; moreover, using all unobserved examples may not largely increase the computational complexity with an appropriate optimization approach.

Besides the above four statements, we have also exploited the most advanced techniques which apply deep learning models for session-based top-N recommendations. Although it also falls into the implicit feedback scenario, the way to generate recommendation is completely different from above works. Specifically, the proposed deep learning model directly estimates the probability distribution for next item(s) instead of computing and ranking the preference scores.

1.4 Thesis Structures and Contributions

This thesis makes a set of contributions regarding performance improvements of recommender systems from implicit feedback. This section mainly discusses the remainder chapters of the thesis along with core ideas and summarized contributions. The thesis is divide into five parts:

- **Part I Introduction:** This part comprises Chapters 1 and 2. It provides the background and related work described in the thesis.
- **Part II Stochastic Gradient Descent with Negative Item Sampling:** This part comprises Chapter 3, 4 and 5. To handle the large-scale unobserved feedback, all algorithms in this part are based on negative sampling techniques with SGD optimization. As for how to effectively distinguish true negatives from unobserved feedback, we introduce LambdaFM (Yuan et al., 2016b) in Chapter 3, BoostFM (Yuan et al., 2017) in Chapter 4, and GeoBPR (Yuan et al., 2016d) in Chapter 5. Detailed motivation and extensive experiments are provided.
- **Part III Batch Gradient Descent without Sampling:** This part comprises Chapter 6. The technical contribution is to show how to efficiently leverage all unobserved examples without resorting to negative sampling methods. Specifically, we propose a generic batch gradient descent method (Yuan et al., 2018b; Xin et al., 2018) that takes advantage of whole training examples for each parameter optimization. To deal with the highly imbalanced data, we propose a simple yet effective weighting scheme for unobserved training data. Moreover, we have also shown that negative sampling based methods easily lead to suboptimal results due to the sampling bias. Empirical evaluations show that batch gradient with all examples significantly outperforms negative sampling based SGD models with comparable training time.
- **Part IV Deep Learning for Session-based Recommendation:** Considering the powerful expressiveness of deep learning models, in Chapter 7

we introduce a convolutional sequence model for session-based recommendation from implicit feedback (Yuan et al., 2018a). The proposed method models conditional distributions of item sequences by the chain rule factorization, and exploits dilated convolutions and residual learning to build the network architecture. Empirical results show that the proposed model achieves both faster training and better recommendation quality.

- **Part V Conclusion:** This part includes only Chapter 8.1 with conclusions.

1.5 Related Publications

The thesis generalizes and builds on the following publications (* denotes equal contribution):

1. **Yuan, Fajie** and Guo, Guibing and Jose, Joemon M and Chen, Long and Yu, Haitao and Zhang, Weinan. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In CIKM 16: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM. (Full paper), **Part II**.

2. **Yuan, Fajie** and Guo, Guibing and Jose, Joemon M and Chen, Long and Yu, Haitao and Zhang, Weinan. Optimizing factorization machines for top-n context-aware recommendations. In WISE 16: International Conference on Web Information Systems Engineering. (Full paper), **Part II**.

3. Guo, Guibing* and Shichang, Ouyang* and **Yuan, Fajie*** and Wang, Xingwei. Approximating Word Ranking and Negative Sampling for Word Embedding. In IJCAI 18: International Joint Conference on Artificial Intelligence. (Full paper), **Part II**.

4. **Yuan, Fajie** and Guo, Guibing and Jose, Joemon M and Chen, Long and Yu, Haitao and Zhang, Weinan. Boostfm: Boosted factorization machines for top-n feature-based recommendation. In IUI 17: Proceedings of the 22nd International Conference on Intelligent User Interfaces, ACM. (Full paper), **Part II**.

5. **Yuan, Fajie** and Jose, Joemon M and Guo, Guibing and Chen, Long and Yu, Haitao and Alkhawaldeh, Rami S. Joint Geo-Spatial Preference and Pairwise Ranking for Point-of-Interest Recommendation. In ICTAI 2016: Tools with Artificial Intelligence. **(Best Student Paper)**, (Full paper), **Part II**.

6. **Yuan, Fajie** and Xin, Xin and He, Xiangnan, Guo, Guibing and Chen, Long and Chua Tat-seng and Jose, Joemon M. f_{BGD}: Learning Embeddings from Positive Unlabeled Data with BGD. In UAI 2018: Association for Uncertainty in Artificial Intelligence. (Full paper), **Part III**.

7. Xin, Xin* and **Yuan, Fajie*** and He, Xiangnan, and Jose, Joemon M. Batch IS NOT Heavy: Learning Word Representations From All Samples. In ACL 2018: Association for Computational Linguistics. (Full paper), **Part III**.

8. **Yuan, Fajie** and Alexandros, Karatzoglou and Ioannis, Arapakis and Jose, Joemon M. and He, Xiangnan. A Simple but Hard-to-Beat Baseline for Session-based Recommendations. arXiv preprint 2018 arXiv:1808.05163 **Part IV**.

Chapter 2

Background of Implicit Recommendation

In this chapter, our main focus is on investigating the item recommendation problem from implicit feedback. Before reviewing related work about implicit recommendation, we first introduce the general background of recommender systems, including the goals, formulations and types. Since our main contribution in this thesis is to develop models for implicit recommendation, we will recapitulate several classic recommendation models, including well-known prediction functions and objective functions for implicit feedback. Finally, we describe the evaluation strategy for implicit feedback recommendation tasks. Specific related work and recommendation models will be further discussed in each contribution chapter.

2.1 Overview on Recommender Systems

2.1.1 Goals and Formulation of Recommender Systems

The goal of a recommender system is defined slightly different in different literature and from different perspectives. From the merchant perspective, the primary goal of a recommender system is to boost product sales so that to increase their profits. By recommending users carefully selected items, recommender systems bring relevant items to matched users, which leads to the increase of sales volume and profits for the business (Aggarwa, 2016). On the other hand, from the customer perspective, the goal of a recommender system should focus on improving the customer experience by, for example, helping them filter out irrelevant items

and find preferred items. Before discussing the goals of recommender systems, we first formulate the recommendation problem by the following two ways:

- Preference value prediction: the recommendation problem is usually formulated as a preference prediction problem (Koren, 2009). The recommendation process is to estimate the relevance score for an observed user-item interaction, and correspondingly, items with higher scores are recommended to the user by priority. As an instance, Netflix awarded a 1 million prize to a developer team in 2009 for a 10% improvement of their company's rating prediction algorithm. In the prediction setting, it is assumed that there are m users and n items, which corresponds to the $m \times n$ matrix. In the matrix, there are a collection of rated items available for each user. The recommender system will then predict the ratings for the remaining unrated items. The problem is also reformulated as a rating completion problem (Aggarwa, 2016) as the recommendation algorithm is designed to estimate the remaining items in the matrix by leveraging the observed ratings.

  It is worth mentioning that preference value prediction is not limited to the rating prediction task. In a practical recommender system, there are various features (e.g., watching time of a video, check-in frequencies of a location, or purchase histories) that can be converted to a user's preference, rather than only user ratings. Moreover, the difference of rating values only indicates whether the user is satisfied or not after the interaction (such as, purchase, watching or listening) with the item, but not explicitly reflects how much he/she like the item before the interaction. In fact, most real-world recommender systems focus more on the prediction accuracy regarding how much the user want to interact with the item, but not the rating after finishing the interaction.

- Item ranking: for real-world recommendation scenarios, the final recommended items are provided to users by a ranked list of potentially matched items. From this perspective, the absolute rating values of recommended items are not important for either users or merchants. For a user, the recommendation is regarded as successful if the matched items returned to them

are at the top (e.g., top-N) positions of the ranked item list. Similarly, for a merchant, the recommender system is more likely to boost sales if matched items are recommended to the user by priority. For this reason, the item ranking task is also referred to as a top-N recommendation problem, which is the ranking formulation of the recommendation problem (Aggarwa, 2016).

It is worth mentioning that although the absolute values of predicted ratings are not important for the item recommendation task, the preference value prediction problem is more general since the estimated scores can also be used for item ranking (i.e., higher scores are ranked higher in the recommendation list.). Moreover, for some related fields such as computational advertising (Zhang, 2016), the recommendation needs to take into account other factors, such as the price of ads. The final recommendation list generated to users are usually based on the multiplication of relevance/preference score and price. In this case, the real-valued relevance score is more important than ranking relations because the multiplication of ranks and price is not accurate. Despite that, many previous work claim that it is easier and more natural to optimize the ranking directly instead of first calculating scores of items and then ranking them by their scores.

In this thesis, we first discuss the way to directly optimize item ranking in Chapter 3, 4 and 5, and then discuss how to generate good recommendations by the way of preference value prediction in Chapter 6. Both options have their own advantages and disadvantages, which are explained in the contribution chapters.

As mentioned, increasing revenues for merchants and improving experience for users are the two primary goals for recommender systems. To achieve this, practical recommender systems usually takes into account several common goals, such as recommendation accuracy, novelty and diversity (Aggarwa, 2016). We briefly introduce the concepts of them since in this thesis our research only focuses on recommendation accuracy.

- Accuracy: recommendation accuracy refers to how much a user likes or enjoys an item. It is usually measured by the relevance between users and returned items. Accuracy or relevance is regarded as the most important goal and evaluation criterion for a recommender system (Castells et al.,

2011). Detailed evaluation methods for recommendation accuracy will be discussed later in this chapter.

- Novelty or serendipity: it refers to the ability of a recommender system generating unusual or novel recommendations which are preferred by the user (Hurley and Zhang, 2011). Novelty or serendipity is a useful metric to remedy the defects of relevance-based recommendations since it often happens that a recommender system always returns a list of relevant items, which however are well-known by the user and thus lack of novelty. Users tend to feel bored when interacting with systems that recommend only known items. From the user perspective, recommending items with some fortunate discoveries inspires their interests and build trust relations with the recommender systems; from the merchant perspective, recommending serendipity items may directly result in the increase of product sales. Moreover, the merchant will have long-term and loyal customers.

- Diversity: In many cases, generating only similar items may not be useful for the users. A recommender system should generate items of different types so that the user may at least like one of these items (Aggarwa, 2016). Note that diversity has different meanings with novelty and serendipity though with similar notions. Specifically, novelty focuses on how different the recommended items are, with respect to "what have been seen before", while diversity focuses on how different the recommended items are, with respect to each other (Vargas and Castells, 2011).

Despite the fact that much literature has recognized that novelty and diversity are also important factors to consider when building practical recommender systems, by far research in diversity and novelty is still not well-established, particularly in the consensus of the evaluation methodologies (Wit, 2008). Hence, this thesis positions itself in the most popular research line, i.e., improving the recommendation accuracy.

2.1.2 Types of Recommender Systems

As mentioned in Chapter 1, recommender systems can be broadly classified into two categories: collaborative filtering and content/context-based recommenda-

tions, according to the way to generate recommendations.

2.1.2.1 Collaborative Filtering Based Recommendations

Collaborative filtering (CF): CF is regarded as the most basic technique used for personalized recommender systems. It is a method of "*making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating)*[1]". In a CF recommender system, it is assumed that a user who has similar tastes or opinions as others may enjoy items preferred by these users. From the technique perspective, CF based recommendation can be grouped into memory-based (Linden et al., 2003) and model-based (Koren et al., 2009) approaches.

Memory-based CF methods, aka neighborhood-based CF, are widely used in early recommendation literature (Sarwar et al., 2001) and industry (e.g., in the Amazon recommender system (Linden et al., 2003)). It can be broadly divided into user-item CF and item-item CF. The common approach for user-item CF is to find users that are similar to the target user by leveraging the similarity of ratings (or interaction frequency), and then recommend items that those similar user liked. By contrast, item-item CF typically first focuses on users who like the particular item, and then recommend other items that those users also liked[2]. For better understanding, they are typically expressed as follows (Linden et al., 2003):

User-item CF: *users who are similar to you also liked/viewed/bought...*

Item-item CF: *users who liked/viewed/bought this also liked/viewed/bought...*

In practice, memory-based CF techniques can be implemented by calculating the distance metric, such as cosine similarity (Linden et al., 2003), Pearson correlation (Sheugh and Alizadeh, 2015) and Jaccard coefficient[3]. For example, in the user-item CF setting, we assume two users, A and B, are represented by two vectors $\mathbf{a}$ and $\mathbf{b}$[4], the cosine similarity is defined as:

$$\text{similarity}(\mathbf{a}, \mathbf{b}) = \cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{\| \mathbf{a} \| \| \mathbf{b} \|} \tag{2.1}$$

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

[2] https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

[3] http://ase.tufts.edu/chemistry/walt/sepa/Activities/jaccardPractice.pdf

[4] $\mathbf{a}$ and $\mathbf{b}$ can be simply created by extracting the corresponding rows or columns from the user-item matrix.

Once the similarity scores are achieved, recommendations can be generated by choosing items that are preferred by the top-N similar users. The main advantages of memory-based recommendations are that they are simple to implement and the generated recommendations can be well explained (Aggarwa, 2016).

In contrast to memory-based CF, model-based CF does not need to explicitly calculate the similarities between users and items. Instead, it usually relies on machine learning and data mining techniques to automatically learn the parameters by certain optimization framework. Examples of model-based methods include but are only limited to latent factor models (aka, embedding models) (Koren et al., 2009), boosting (Chen and Guestrin, 2016), tree models (Loh, 2011), SVM (Herbrich et al., 1999) and neural network models (He et al., 2017). Among these methods, factorization models (such as matrix factorization (Koren et al., 2009) and factorization machines (Rendle, 2012)) and neural network models (He et al., 2017) are most successful in recent literature and are also the focus of this thesis. Compared with the memory-based methods, model-based CF techniques are usually built based on a low-dimensional models (e.g., factorization techniques). As a result, model-based models take less memory since they do not need to store the original rating matrix; moreover, they are usually much faster in the preprocessing phase as the quadratic complexity for calculating similarity between users and items are omitted (Aggarwa, 2016). Another advantage is the regularization strategy which helps models avoid the overfitting problem (Aggarwa, 2016). In this thesis, all our proposed algorithms are model-based. Full details will be further discussed later in this chapter and also in the following contribution chapters.

2.1.2.2 Content/Context-Based Recommendations

The basic collaborative filtering based recommendation only consider the relations of users and items without leveraging other information. However, in many practical recommender systems, items and users are often characterised by a set of predefined features (Wit, 2008). For example, in a news recommender system, the content (e.g., words, title, author, and genre) of the news contains significant features to describe its properties. In this scenario, the basic CF models may lose important information, leading to suboptimal recommendations. In Chapter 3

and 4, we have shown that in a music recommender system, the features of music tracks, i.e., the album and artist information, largely influence the accuracy of recommendation models. The content information can be obtained directly or extracted from the recommender system. Recommendations based on the content information of items is usually referred to as content-based recommendations. Moreover, for a real-world recommender systems, there are some additional context information for users, such as the location, time, social friends, weather and seasons, which also has an effect on users' preference and decisions making. Still using the same example, a music recommender system provides different recommendations based on the time of the day. In fact, it is quite possible a user's preference for a music may change in different time of a day[1]. Recommendation process based on context information of users is often referred to as context-based or context-aware recommendations. For clarity, we refer user-related features as context (including user's profile) and item-related features as content in this thesis. The main advantage of content/context-based recommendation is to alleviate the problem of data sparsity. It usually performs well when there are enough user and item feature information. However, one drawback is that the content and context information is often missing for many users and items in large-scale real-world recommender systems. In practice, recommender systems usually combine both collaborative filtering and content/context information (De Campos et al., 2010), referred to as the hybrid recommendation. In our thesis, we design recommendation models which can be used in both basic collaborative filtering and content/context-based settings. For example, we evaluate the performance of LambdaFM (Yuan et al., 2016b,a), BoostFM (Yuan et al., 2017), GeoBPR (Yuan et al., 2016d) and f_{BGD} (Yuan et al., 2018b) with both basic collaborative filtering baselines and content/context-based models. Note that the neural network model of NextItNet proposed in Chapter 7 is developed without considering the content/context-information.

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

2.2 Overview on Implicit Recommendation

As has been mentioned in Chapter 1, recommender systems can also be classified according to the type of observed feedback. Most early recommendation literature focus on explicit feedback based on users' ratings (Koren et al., 2009; Karatzoglou et al., 2010), while implicit feedback, such as views, clicks and purchases, is more pervasive and can be collected in a much cheaper way than user ratings (Rendle and Freudenthaler, 2014) since users do not need to explicitly express their preference. In recent years, implicit feedback has attracted more and more attention than explicit feedback in the recommender system domain (Hu et al., 2008; Pan et al., 2008; Rendle et al., 2009b; Pan and Chen, 2013; Zhao et al., 2014; Rendle and Freudenthaler, 2014; Yuan et al., 2016d,b, 2017; He et al., 2016c; Bayer et al., 2017). In this section, we will review related contributions regarding item recommendations from implicit feedback.

The most important characteristics of implicit feedback based recommendation are: (1) negative and missing (unobserved) feedback are mixed together; (2) unobserved feedback has a much larger scale than positive feedback; and thus (3) the training data has huge sparsity. It is computationally expensive to leverage all unobserved feedback for designing a recommendation algorithm. To solve these problems, a variety of recommendation models have been proposed.

To deal with these issues, several solutions were proposed. A typical solution is to treat all unobserved data as negative example by Pan et al. (2008). Clearly, this strategy are not optimal since many unobserved items could be positive if the user knows them. Hence, treating all of unobserved data as negative may mislead the learning model towards an incorrect optimization direction. The other way is that we treat all unobserved data as unknown, and feed the recommendation model with only positive data. By this method, the predictions on all data will be positive values since only positive data is available for training; moreover, the recommendation may result in poor prediction accuracy since a large amount of useful negative data is missing during training. The above extreme strategies are referred to as AMAN (all missing as negative) and AMAU (all missing as unknown) (Pan et al., 2008).

To overcome the two extremes, Pan et al. (2008) proposed to balance the

extent of treating unobserved data as negative. They proposed two solutions, namely weighting and sampling, which allow to tune the tradeoff when using negative examples. Specifically, they proposed a weighted low-rank approximation technique following Srebro and Jaakkola (2003), which solves a generic problem with positive data as "1" and unobserved data as "0". Furthermore, they improved the original model by designing three types of negative weighing schemes, namely uniform, user-oriented and item-oriented weighting schemes. Specifically, they first assign a constant value as the weights for all negative examples, which corresponds to the AMAN assumption. As just mentioned, the approach easily results in suboptimal recommendation quality since not all unobserved data are negative. To improve the uniform weight, they proposed the second weighting scheme, i.e., user-oriented weight. The assumption is that if a user has shown more positive observations, it is more likely that he/she does not like other items. That is, the unobserved data for this user has more chances to be real negative data. The third weight is item-oriented, which assumes that if an item has seldom been chosen as positive examples, the unobserved data for this item is negative with higher probability. However, during evaluation the authors found that the item-oriented weighting scheme is worse than the uniform weight. The main reason is probably because they made an opposite assumption since less-be-chosen (i.e., unpopular) items may have lower probability to be true negative than the popular items. This is intuitively correct because the popular item that is not chosen by the user probably suggests that he/she does not like it considering that the user may know it well due to its popularity. Following the same intuition of weighting schemes, they also investigate three sampling strategies, which can be regarded as the first work about negative sampling in the recommender system field. Their empirical results show that both weighting and sampling methods outperform baselines (i.e., AMAN and AMAU).

Another well-known work for implicit feedback recommendation work is proposed by Hu et al. (2008), referred to as Weighted Regularized Matrix Factorization (WRMF). The work has the same intuition with Pan et al. (2008) by optimizing a weighted least square regression function. The main difference is that Hu et al. (2008) designed the weight only for the positive data while all negative data is treated equally. In addition, because of computing the closed-

form equation, the time complexity of the standard optimization is cubic in terms of the dimensions of latent factors. To solve the efficiency issue, a fast matrix factorization model by leveraging element-wise alternative least square (ALS) learner was proposed by He et al. (2016c). The time complexity of the efficient ALS optimizer (eALS) has reduced from $O(k^3)$ to $O(k^2)$, where k is the embedding dimension. Although eALS has achieved important speed-up, it can only be used for the basic collaborative filtering setting with only user id and item id as input features. Motivated by this, a following work by Bayer et al. (2017) has presented a generic eALS model, referred to as iCD, which can be applied to any "k-separate" model, such as matrix factorization (Koren et al., 2009), factorization machines (Rendle, 2010) and some tensor factorization models (Xiong et al., 2010). iCD can not only be used for basic collaborative filtering scenario, but also in context-aware item recommendations. While effective, the proposed generic eALS models have not been compared with the state-of-the-art baselines. As a result, its real performance in item recommendation task is unknown. In addition, iCD is optimized by the Newton method, which relies on the second-order derivative and is sensitive to parameter initialization and regularization terms.

Another line of research for implicit recommendation is based on learning-to-rank (LtR) models (Liu et al., 2009). This is because the implicit item recommendation task is essentially to solve a ranking problem, i.e., ranking the relevant or preferred items for the user at the top of the candidate list. The seminal work of the LtR model for implicit feedback scenario is called Bayesian personalized ranking (BPR) by Rendle et al. (2009b), which is a pairwise LtR algorithm. The basic idea of BPR is to make sure that the observed items should be ranked higher than the unobserved ones under the rule of Bayesian maximum a posteriori probability (MAP) estimate. BPR optimization is generic and not limited to the basic matrix factorization model. For example, many works proposed to use BPR to optimize tensor factorization models (Rendle and Schmidt-Thieme, 2010), pairwise interaction tensor factorization (Rendle and Schmidt-Thieme, 2010) machines, factorization machines (Rendle, 2010), and neural networks (Niu et al., 2018). Moreover, there are also some works (Pan and Chen, 2013; Zhao et al., 2014; Rendle and Freudenthaler, 2014; Yuan et al., 2016b) that claim the original BPR assumption is suboptimal in specific implicit feedback tasks. For exam-

ple, Pan and Chen (2013) argued that the original BPR assumption of the joint likelihood of pairwise preferences of two users may not be independent of each other, and further a user may potentially like an unobserved item to an observed one. To deal with this, the authors made a new assumption and introduced the group preference by incorporating more interactions among users. They named the algorithm as group Bayesian personalized ranking (GBPR). The basic idea of GBPR is to replace the individual pairwise relationship with a group relationship that involves group preference. After GBPR, social BPR (Zhao et al., 2014) is proposed that incorporates the social preference. The assumption of social BPR is that a user's preference to a positive item should be higher than that of the unobserved items which are positive to his/her social friends, which are further higher than that of the unobserved items without any interaction by his/her friends. There are also work (Rendle and Freudenthaler, 2014) claiming that the uniform sampler in the original BPR is suboptimal since the item distribution in real-word datasets are tailed. As a result, most randomly sampled unobserved items fall into the long tail. Rendle and Freudenthaler (2014) pointed out that the unobserved items that are into the long tail are not informative and contribute less on the parameter update by stochastic gradient descent (SGD) optimization. To address the problem, they proposed an adaptive sampling that dynamically draws unobserved items with smaller ranks (or higher scores). They further showed that BPR with the adaptive sampler converges much faster than that with the uniform sampler. Interestingly, they found that the popularity-based sampler hurts the recommendation accuracy although it converges much faster. While in our evaluation we empirically found that BPR with the popularity sampler can significantly improve the basic uniform sampler (Yuan et al., 2016b). The reason may be because in Rendle and Freudenthaler (2014) the authors oversampled only a small portion of very popular unobserved items while most relatively popular items are missing. Losing important negative examples will easily result in poor performance. Our finding is consistent with that in the word embedding task[1].

We noticed that there were also several listwise based recommendation models

[1] We observed that the adaptive sampling (Rendle and Freudenthaler, 2014) idea can be used beyond the recommender system domain. For example, we have applied the same approach for the word embedding task and achieved better embedding vectors than the original popularity-based sampler (See Chen et al. (2017)).

which can be used for the implicit feedback scenario, such as (Shi et al., 2012b,a). Although listwise methods directly optimize the top-N measures, most methods empirically do not yield better recommendations than the pairwise BPR model (Shi et al., 2014). The is because most top-N measures are either non-differential or non-continuous. Approximating these ranking measures or proposing bound theories typically cannot guarantee the ranking accuracy. For example, Shi et al. (2014) showed that BPR optimization is better than the listwise optimization with the same prediction function. Hence, in this thesis, most of the proposed recommendation models are either pairwise[1] or pointwise.

2.3 Implicit Feedback Model Overview

A recommendation model typically consists of two modules: a prediction function and a loss function. Considering that features in recommender systems are usually very sparse, prediction functions are reqired to have the capacity in dealing with sparsity. One of the most popular prediction functions are factorization (or embedding) based models, which are superior to classic nearest-neighbor techniques (Koren et al., 2009). Recently, deep learning models have attracted much attention in recommender systems domain. Compared with shallow factorization models, deep learning models are more expressive, especially for capturing the non-linear relations between the low-rank latent vectors of users and items. In terms of loss functions, they are generally divided into three categories, pointwise, pairwise and listwise approach. Among these loss functions, pairwise and pointwise approach have gained more popularity. In the following, we will first recapitulate well-known prediction models including basic factorization models and complex deep learning models, and then introduce two ways of model training (i.e., pointwise and pairwise) alongside with the sampling strategies.

2.3.1 Factorization Models

In recent decade, latent factor models, aka embedding models, have become the most successful model in collaborative filtering based recommendation. These models have been comprehensively studied for various recommendation tasks.

[1] It is worth mentioning that the LambdaFM model proposed in Chapter 3 is implemented by the pairwise method although it is formulated by the listwise motivation.

In this section, we will systematically review several representive factorization models.

2.3.1.1 Basic Matrix Factorization (Koren et al., 2009)

Matrix factorization (MF) is regarded as the most widely used latent factor model in recent years (Koren et al., 2009). MF-based models are not only used in recommender system domain, but also widely used in natural language process (Pennington et al., 2014) and computer vision (Weston et al., 2011) domains. The factorized low-rank matrix is also referred to as embedding matrix. The basic idea of matrix factorization model is to map both users and items to a joint latent factor space with dimension k. The user-item interactions are typically modeled as dot product in the mapped space. Following the definition in Koren et al. (2009), let $\mathbf{q}_i \in \mathbb{R}^k$ be the vector that contains latent features of item i, and on similar lines, $\mathbf{p}_u \in \mathbb{R}^k$ be the vector for user u. Specifically, for a given item i, $\mathbf{q}_i$ represents the extent to which the item possesses those features, positive or negative, large or small; for a given user u, $\mathbf{p}_u$ describes the components of user's profile, i.e., how much the user is interested in the corresponding factors. The dot product rating function $\mathbf{q}_i^T \mathbf{p}_u$ captures the interaction between u and i — the overall preference u has on i. Let $\hat{y}_{ui}$ be the rating function, we have

$$\hat{y}_{ui} = \mathbf{q}_i^T \mathbf{p}_u = \sum_{f=1}^{k} p_{uf} q_{if} \tag{2.2}$$

where higher value of $\hat{y}_{ui}$ denotes user u and item i have a more relevant match. Previous studies (Gogna and Majumdar, 2015) also showed that actual ratings are not only a simple dot product operation between the user and item latent vectors but also contains bias. For example, users who usually rate higher than the average score have a positive bias. Likewise, items that are popular tend to be rated higher by most users than long tailed items also have a positive bias. With the bias terms, the rating equation can be defined as

$$\hat{y}_{ui} = \mathbf{q}_i^T \mathbf{p}_u + b_u + b_i + b \tag{2.3}$$

where b_u, b_i and b are user bias, item bias and the average rating of all observed ratings. For implicit feedback settings, b_u and y can be safely removed since the values of them do not influence the overall rank of items for the same user.

2.3.1.2 SVD++ (Koren and Bell, 2015)

SVD++ is an enhanced factorization model by considering users' additional information as opposed to $\mathbf{p}_u$ that only includes the latent features of current user. According to (Koren and Bell, 2015), SVD++ offers accuracy superior to basic MF model, particularly in cases where independent implicit feedback is missing since one can capture an important signal by taking account of which items users have rated, regardless of their rating value.

Following the definition in Koren and Bell (2015), in the SVD++ model a second set of item factors is added into the user latent vector, relating each item i to a factor $y_i \in \mathbb{R}^k$. The additional item factors are utilized to describe users based on the set of items that they have previously rated. The model is defined by

$$\hat{y}_{ui} = b_u + b_i + b + \mathbf{q}_i^T(\mathbf{p}_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} z_j) \tag{2.4}$$

where $R(u)$ denotes the set of items rated by user u.

Different from the basic MF model, in SVD++, a user is described by two components, i.e., $\mathbf{p}_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} z_j$. $\mathbf{p}_u$ is learned from the explicit rating data, while the additional term $\sum_{j \in R(u)} z_j$ represents the effect of implicit feedback. Various SVD++ versions can be designed by the similar way to construct additional user implicit feedback. For example, if the user has implicit preference to the items denoted by $R^1(u)$, and also another type of implicit feedback to the items denoted by $R^2(u)$, we could have a more expressive model (Koren and Bell, 2015)

$$\hat{y}_{ui} = b_u + b_i + b + \mathbf{q}_i^T(\mathbf{p}_u + |R^1(u)|^{-\frac{1}{2}} \sum_{j \in R^1(u)} z_j^{(1)} + |R^2(u)|^{-\frac{1}{2}} \sum_{j \in R^2(u)} z_j^{(2)}) \tag{2.5}$$

By this way, the preference of each source of implicit feedback can be automatically learned by the algorithm by updating respective parameters.

2.3.1.3 SVDFeature (Chen et al., 2012)

SVDFeature is a machine learning tookit designed for feature-based collaborative filtering. Its feature-based setting allows the model to incorporate various side information (e.g., social relations, users' context and item metadata) for both user and items with different input data. The original SVDFeature prediction

function models three factors as a feature vector. Following (Chen et al., 2012), it is defined as

$$\hat{y}_{ui} = \sum_{d=1}^{s} \gamma_d b_d^g + \sum_{d=1}^{n_1} \alpha_d b_d^u + \sum_{d=1}^{n_2} \beta_d b_d^i + (\sum_{d=1}^{n_1} \mathbf{p}_d \alpha_d)^T (\sum_{d=1}^{n_2} \mathbf{q}_d \beta_d) \tag{2.6}$$

where the model parameter $\Theta = \{b^g, b^u, b^i, \mathbf{p}, \mathbf{q}\}$. $\mathbf{p}_d \in \mathbb{R}^k$ and $\mathbf{q}_d \in \mathbb{R}^k$ are k dimensional latent factors associated with each feature. b_d^u, b_d^i and b_d^g are bias terms for user, item and global features (i.e., α_d, β_d and γ_d respectively).

According to Chen et al. (2012), SVDFeature is a generic prediction function and can reduce to basic matrix factorization if the feature vector only includes user and item features. Moreover, it can mimic several state-of-the-art factorization or embedding models, and can be applied not only in recommender systems but also in other fields such as word embedding with word prior knowledge. According to Eq. (2.6), we can see SVDFeature only models interactions between user- and item- field variables without considering the interactions in each field.

2.3.1.4 Factorization Machines (Rendle, 2010)

Factorization Machines (FM) is a generic predictor that combines the merits of Support Vector Machines (SVM) with factorization models. However, compared with SVM, FM is able to deal with very sparse data with any type of real valued feature vector using factorized parameters. More detailed relations with SVM has been shown in the original paper (Rendle, 2010). In addition, it can mimic most factorization models (e.g., MF, SVD++, timeSVD++ (Koren, 2010), FPMC (Xiong et al., 2010), PITF (Rendle and Schmidt-Thieme, 2010)) just by feature engineering. In contrast to SVDFeature, it takes into account of all pairwise interactions between variables. According to Rendle (2010), the model equation for a factorization machine of degree $m = 2$ is defined as

$$\hat{y}(\mathbf{x}) = \underbrace{w_0 + \sum_{d=1}^{n} w_d x_d}_{\text{linear}} + \underbrace{\sum_{d=1}^{n} \sum_{d'=d+1}^{n} \langle \mathbf{v}_d, \mathbf{v}'_d \rangle x_d x'_d}_{\text{polynomial}} \tag{2.7}$$

where x_d is d-th element in the sparse feature vector $\mathbf{x} = (\mathbf{x}_c, \mathbf{x}_i)$. $\mathbf{x}_c$ and $\mathbf{x}_i$ are regarded as the latent/embedding factors of user-field (or context)[1] and item-field (or item) respectively. For example, in the feature-based recommender system

[1] A user-field refers to the user-related context, such as location, time and social friends. In the following chapters, we use user-field (e.g., in Chapter 6) and context (e.g., in Chapter 3 & 4) interchangeably.

setting, $\mathbf{x}$ can be expressed as follows:

$$\mathbf{x} = \underbrace{(\overbrace{0,...,1,...,0}^{\text{users}},\overbrace{0,0.5,...,0.5,0}^{\text{friends}},\overbrace{0,1}^{\text{location}},\overbrace{0,1,0}^{\text{time}},}_{\mathbf{x}_c\text{:user-field (or context)}}\underbrace{\overbrace{0,...,1,...,0}^{\text{music tracks}},\overbrace{0,...,1,...,0}^{\text{artists}},\overbrace{0,...,1,...,0}^{\text{albums}})}_{\mathbf{x}_i\text{:item-field (or item)}}$$

The model parameters Θ that have to be estimated are $w_0 \in \mathbb{R}$,$\mathbf{w} \in \mathbb{R}^n$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$, and $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors of size k:

$$\langle \mathbf{v}_d, \mathbf{v}_{d'} \rangle = \sum_{f=1}^{k} v_{d,f} \cdot v_{d',f} \tag{2.8}$$

The linear part of FM contains unary interactions of each input variables x_k with the target, which is identical to a linear regression model. The polynomial part models the interaction between the d-th and d'-th variables, which distinguishes itself from standard polynomial regression by using a factorized parametrization. The authors have proved that FM can be computed in linear complexity $O(kn)$ as Eq. (2.7) can be reformulated as

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{d=1}^{n} w_d x_d + \frac{1}{2} \sum_{f=1}^{k} \left((\sum_{d=1}^{n} v_{d,f} x_d)^2 - \sum_{d=1}^{n} v_{d,f}^2 x_d^2 \right) \tag{2.9}$$

In collaborative filtering scenarios, most elements x_k in a vector $\mathbf{x}$ are zero. For example, let $N(\mathbf{x})$ be the number of non-zero elements in the feature vector $\mathbf{x}$ and $\overline{N(\mathbf{x})}$ be the average number of non-zero elements in all vectors. We can see that $\overline{N(\boldsymbol{x})} \ll n$ under huge sparsity, i.e., the complexity becomes $O(k\overline{N(\mathbf{x})})$ in the sparse setting.

2.3.1.5 Tucker Decomposition (Tucker, 1966)

Tucker Decomposition (TD) have also been widely used in feature-based recommendation task, e.g., tag recommendation (Rendle et al., 2009a), by computing all interactions between the factorization matrices of users, items and tags. The prediction function of tensor model is computed by multiplying the three feature matrices to the core tensor, given by

$$Y = \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{2.10}$$

where $\hat{C}$ is the core tensor, and $\hat{U}$, $\hat{I}$ and $\hat{T}$ are the feature matrices for users, items and tags. $\times x$ denotes the tensor product that multiples a matrix on dimension x with a tensor. The model parameters Θ are as follows

$$\hat{C} \in \mathbb{R}^{k_U \times k_I \times k_T}, \hat{U} \in \mathbb{R}^{|U| \times k_U}, \hat{I} \in \mathbb{R}^{|I| \times k_I}, \hat{T} \in \mathbb{R}^{|T| \times k_T}$$

where k_U, k_I and k_T denote the dimension of the low-rank approximation. According to Rendle et al. (2009a), the prediction function $\hat{y}_{uit}$ of tensor factorization with given feature matrices and the core tensor is defined by

$$\hat{y}_{uit} = \sum_{f_1=1}^{k_U} \sum_{f_2=1}^{k_I} \sum_{f_3=1}^{k_T} c_{f_1,f_2,f_3} \hat{u}_{u,f_1} \hat{i}_{i,f_2} \hat{t}_{t,f_3} \tag{2.11}$$

It is worth noticing that TD is much more computationally expensive than matrix factorization or factorization machines, which requires $O(k_U k_I k_T)$ complexity.

2.3.2 Deep Learning Models

Although factorization models have achieved huge success during the past decade, the dot product structure of which have limitations in modeling complex (or non-linear) interactions between users and items (He et al., 2017). The neural network with multiple nonlinear layers have been proven to be capable of approximating any continuous function. There are some recent advances that applied deep neural networks (DNN) for recommendation tasks and showed state-of-the-art performance than classic factorization models. More specifically, He et al. (2017) have proposed a neural network based collaborative filtering framework, referred to as NCF, by combining classic matrix factorization and multi-layer perceptron. NCF has demonstrated significant improvement in contrast with MF model, particularly with more hidden layers. A following work by He et al. (2017) designed Neural Factorization Machine (NFM) on top of the embedding vector of FM by stacking additional neural layers. Likewise, NFM is proven to be more expressive than basic FM model with only one hidden layer. In the following, we shortly recapitulate the two models.

2.3.2.1 Neural Collaborative Filtering (He et al., 2017)

The basic idea of NCF model is to replace the matrix factorization (or dot product) function with a neural architecture so that it can learn and approximate any distribution in the data. The main structure NCF model is composed of two parts: a generalized matrix factorization (GMF) and a multi-layer perceptron (MLP). Specifically, GMF can be regarded as a generalized and extended MF model since MF is only a special case of GMF. The prediction function of GMF

is defined as

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i)) \tag{2.12}$$

where σ and $\mathbf{h}$ denote the activation function (i.e., sigmoid function in the paper) and edge weights of the output layer respectively. Clearly, let σ be an identity function and $\mathbf{h}$ be a constant vector of 1, GMF is reduced to the MF model. However, if we automatically learn $\mathbf{h}$ by training and use non-linear function for σ, it will generalize MF to a non-linear setting according to He et al. (2017).

The other component MLP is a concatenation of user and item latent vectors. This design has been widely used in various deep learning work. However, a simple vector concatenation does not have enough expressiveness to model the interactions of user and item latent features. To enhance the model, the authors add more hidden layers on the concatenation vector by adding MLP to learn the interactions. According to He et al. (2017), the MLP model is defined as

$$\begin{aligned} \mathbf{z}_1 &= \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix} \\ \psi_2(\mathbf{z}_1) &= \alpha_2(\mathbf{W}_2^T\mathbf{z}_1 + \mathbf{b}_2) \\ &\dots \\ \psi_L(\mathbf{y}_{L-1}) &= \alpha_L(\mathbf{W}_L^T\mathbf{z}_{L-1} + \mathbf{b}_L), \\ \hat{y}_{ui} &= \sigma(\mathbf{h}^T\psi_L(\mathbf{y}_{L-1})) \end{aligned} \tag{2.13}$$

where W_x, b_x and α_x represent the weight matrix, bias vector and activation function for x-th layer respectively. The final NCF model is a combination of GMF and MLP with a one-layer MLP, which is formulated as

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T\alpha(\mathbf{p}_u \odot \mathbf{q}_i + \mathbf{W}\begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix} + \mathrm{b})) \tag{2.14}$$

Note that in the original paper, the authors argue that GMF and MLP share the same embedding layer might limit the performance of the combined model and thus their final training model learns separate embeddings for GMF and MLP.

2.3.2.2 Neural Factorization Machines (He and Chua, 2017)

The NCF model has demonstrated stronger performance than state-of-the-art factorization models due to the non-linear property, while NCF is tailed for basic collaborative filtering with only user and item features. Intuitively, it is natural

to apply the NCF strategy for a feature-based factorization model such as Factorization Machines (FM). In the following work of He et al. (2017), they present a neural factorization machine (NFM) model, which increase the non-linearity of the shallow FM model by stacking more neural layers. With this design, the original FM model can be seen as a special case of NFM. The most important module of NFM is a Bilinear Interaction (Bi-Interaction) pooling operation, which is designed inspired by the efficiency transformation of the original FM model.

$$f_{BI}(\mathbf{V}_x) = \sum_{d=1}^{n} \sum_{d'=d+1}^{n} \langle \mathbf{v}_d, \mathbf{v}'_d \rangle x_d x'_d = \frac{1}{2}\Big((\sum_{d=1}^{n} \mathbf{v}_d x_d)^2 - \sum_{d=1}^{n} \mathbf{v}_d^2 x_d^2 \Big) \tag{2.15}$$

where $\mathbf{V}_x = \{x_d \mathbf{v}_d\}$. The main difference from the original FM is that original FM directly computes the final score by the above transformation while NFM will keep the intermediate result which is the embedding vector. The NFM model is built on the Bi-Interaction pooling layer with a stack of fully connected layers, which are capable of learning higher-order interactions between features (Shan et al., 2016). The prediction function NFM model is given by:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{d=1}^{n} w_d x_d + \mathbf{h}^T \sigma_L(\mathbf{W}_L(...\sigma_1 \mathbf{W}_1 f_{BI}(\mathbf{V}_x) + \mathbf{b}_1)...) + \mathbf{b}_L \tag{2.16}$$

where $\mathbf{W}_l$,$\mathbf{b}_l$ and σ_l are the weight matrix, bias vector, activation function for the l-th layer respectively; $\mathbf{h}$ is the neuron weights of the final prediction layer. The model parameters of NFM are $\Theta = \{w_0, \{w_d, \mathbf{v}_d\}, \mathbf{h}, \{\mathbf{W}_l, \mathbf{b}_l\}\}$.

2.3.3 Objective Functions with Negative Sampling

The learning of recommendation models can be mathematically described as an optimization problem, the goal of which is to find the best or optimal solution for the task. To find the optimal resolution, a typical way is to develop the measurement for the quality of the solution, which is achieved by objective/loss function. Two most well-known objective functions in the recommender system domain are pointwise and pairwise methods. Specifically, the pointwise loss function is typically designed to optimize the least square loss or logistic regression loss, the goal of which is to minimize the errors between the ground truth score and estimated score by prediction function. By contrast, the pairwise loss does not directly optimize the predicted score but instead takes into account the rank relations of positive and item examples, and penalizes the model if the rank re-

lation is incorrect. In this thesis, Chapter 3, 4 and 5 are based on the pairwise learning-to-rank optimization and negative sampling, while Chapter 6 is based on the pointwise regression method without sampling (i.e., with full sampling).

2.3.3.1 Pointwise Loss with Negative Sampling

Pointwise loss function concentrates on a singe example (i.e., a user-item pair) at a time. The basic idea is to deal with a single example and train it with a regressor/classifier to predict the relevance score that the user has for the item. There are two commonly used pointwise losses, namely, a square loss for regression (Pan et al., 2008) and a logistic loss for classification (Mikolov et al., 2013).

Weighted square loss function is the sum of squared difference between target values and predicted values with a weighting coefficient.

$$MSE = J(\Theta) = \sum_{(u,i)\in U\times I} \alpha_{ui}(\hat{y}_{ui} - y_{ui})^2 \tag{2.17}$$

where U and I are the set of users and items. α_{ui} is the weight for the user-item pair, $\hat{y}_{ui}$ and y_{ui} are the predicted score and ground truth score. By separating the loss function into positive and unobserved negative examples, it can be rewritten as

$$J(\Theta) = \underbrace{\sum_{(u,i)\in S} \alpha^+_{ui}(y^+_{ui} - \hat{y}_{ui})^2}_{J_P(\Theta)} + \underbrace{\sum_{(u,i)\in (U\times I)\setminus S} \alpha^-_{ui}(y^-_{ui} - \hat{y}_{ui})^2}_{J_M(\Theta)} \tag{2.18}$$

where S is the set of positive examples, $J_P(\Theta)$ and $J_M(\Theta)$ are training loss errors for positive and unobserved examples respectively; y^+_{ui} and y^-_{ui} are ground truth score for positive and unobserved data. It is worth noticing that the computational complexity of $J_M(\Theta)$ is almost in $O(|U||I|)$ since the number of positive items a user has interacted is much less than that of unobserved items. To deal with such a large number of unobserved examples, most recommendation models rely on negative sampling (NS) and stochastic gradient descent (SGD) optimization for efficient optimization. The basic idea of negative sampling is to select a small portion of unobserved examples, for example K percentage of them, as negative for model training to avoid all examples based parameter updating.

$$J(\Theta) = \underbrace{\sum_{(u,i)\in S} \alpha^+_{ui}(y^+_{ui} - \hat{y}_{ui})^2}_{J_P(\Theta)} + \underbrace{\sum_{(u,i)\in K((U\times I)\setminus S)} \alpha^-_{ui}(y^-_{ui} - \hat{y}_{ui})^2}_{J'_M(\Theta)} \tag{2.19}$$

The most commonly used negative sampling is based on the uniform sampling.

Logistic loss is also a widely used pointwise method in the implicit recommendation task. The loss function is also used by the Skip-gram model (Mikolov et al., 2013) in the word embedding task. The sampling based loss function is defined as

$$J(\Theta) = \underbrace{-\sum_{(u,i)\in P} \log \hat{y}_{ui}}_{J_p(\Theta)} \underbrace{- \sum_{(u,i)\in (K(U\times I)\setminus S)} \log \hat{y}_{ui}(1-\hat{y}_{ui})}_{J_M(\Theta)} \tag{2.20}$$

The intuition behind this loss function is to maximize the score of positive examples while in the same time minimize the score of negative examples. Both the square loss and the logistic loss offer reasonable recommendations. Theoretically, the logistic loss may offer better performance than the square one since item recommendation is actually a classification task rather than a regression one. This is because users in practice do not care the prediction scores for items but are more interested in the rank order of items, especially the top ranked items.

2.3.3.2 Pairwise Loss with Negative Sampling

The most well-known pairwise loss function[1] is the Bayesian Personalized Ranking (BPR) (aka cross-entropy) loss by Rendle et al. (2009b). BPR is the first learning-to-rank (LtR) algorithm in the recommender system domain and has achieved huge success. In contrast with the pointwise loss function, pairwise loss directly optimizes its model parameters for ranking instead of real-valued scores. It is worth noticing that the BPR loss is essentially the same loss as RankNet (Burges et al., 2005), while RankNet learns the shallow neural network as a scoring function which is non-personalized. The basic idea of the BPR model is to maximize the cross-entropy loss so that it can maximize the margin between prediction scores of positive examples and negative examples. The BPR loss function is defined as

$$J(\Theta) = \ln \prod_{(u,i,j)\in D_s} \sigma(\hat{y}_{uij}) p(\Theta) = \sum_{(u,i,j)\in D_s} \ln \sigma(\hat{y}_{uij}) - \lambda_\Theta ||\Theta||^2 \tag{2.21}$$

let I_u^+ denote the set of positive items user u has interacted, then $D_s = \{(u,i,j)|i \in I_u^+ \wedge j \in I \setminus I_u^+\}$ represents the set of all pairwise preferences $D_s \subseteq U \times I \times I$, σ

[1] We have compared the BPR loss with serveral well-known pairwise loss functions for the implicit recommendation task in Chapter 3.

and λ_Θ are the logistic sigmoid and regularization parameters. Similar to the the pointwise loss function, the computational cost of the standard BPR loss with full gradient is not feasible since it requires $O(|S||I|)$ training triples, where $|S|$ is the number of all positive examples. Moreover, the optimization for the loss with full gradient descent leads to slow convergence. To solve above-mentioned issues, Rendle et al. (2009b) proposed a uniform sampling strategy for unobserved examples with the SGD optimization. Specifically, instead of using all unobserved examples, they randomly choose the same number of unobserved example as negative to construct the (u, i, j) triple. In this case, it only required $|S|$ training triples, and the computational cost has been largely reduced.

Empirically, with the same negative sampling setting, the pairwise BPR loss usually offers better recommendations than the pointwise approach (Yuan et al., 2018b). This is because the item recommendation task from implicit feedback is actually a ranking task rather than a regression one. The optimal results in terms of RMSE may lead to suboptimal results for ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) (McFee and Lanckriet, 2010) and Mean Reciprocal Rank (MRR) (Shi et al., 2012b). However, an advantage of pointwise loss function is the flexible sampling ratio for negative examples. While the pairwise loss function can only construct training pairs by one sampled negative example and a positive one (He et al., 2017). In Chapter 6, we have extensively studied the effectiveness on item recommendation by changing the sampling distribution and size of negative examples.

It is worth mentioning that the above discussed pairwise and pointwise loss functions are generic and can be applied to various prediction functions introduced in Section 2.3.1 and 2.3.2 as long as these functions has continuous first order derivative.

2.4 Evaluation of Implicit Recommendation

2.4.1 Implicit Feedback Datasets

In this thesis, we evaluate all our recommendation models on real-world datasets, such as location recommendation dataset in Yelp[1], music recommendation dataset

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

in Yahoo![1] and Lastfm music[2]. Each dataset includes at least a number of user IDs and item IDs. Each entry in the dataset contains an observed user-item pair. Note explicit rating data can also be used for implicit feedback based item recommendation. The common approach is to translate rating data as a binary data. For example, we can assume that any ratings above 3 correspond to the positive feedback, and unobserved feedback and ratings below or equal to 3 is negative feedback. The rating value 3 is a threshold to be set according to the actual application. Some datasets may contain user contexts and item metadata (see $\mathbf{x}$ in section 2.3.1.3), which can be used for context-aware recommendations. In each dataset, all observed feedback is positive, whereas we usually do not store unobserved feedback as it can be automatically calculated. It is worth mentioning that in this thesis we do not target the cold start problem. To this end, we usually filter out users and items with a small number interactions (e.g., less than 5) (Rendle et al., 2009b). The detailed statistics of each dataset are shown in each contribution chapter.

2.4.2 Evaluation Protocols

For evaluation, the recommendation dataset will first be split into training and testing sets, such as in Rendle et al. (2009b). The recommendation models are learned on the training set, while evaluated on the testing set. For example, we can use 80% of the dataset for training, while the remaining 20% is for testing. The most commonly used evaluation protocols are 5-fold cross validation (Yuan et al., 2016b) and leave-one-out evaluation (Rendle et al., 2009b). All high-parameters are manually tuned based on either the training loss or a validation set (e.g., in Chapter 7). Specifically, we can search the optimal value for a specific hyper-parameter while fixing others, following Rendle and Schmidt-Thieme (2010). In this thesis, all evaluation is based on the offline protocol (Levy, 2013) (i.e., based on unchanged datasets). The detailed evaluation strategies are given in each chapter for corresponding models.

[1] https://uk.yahoo.com/
[2] https://www.last.fm

2.4.3 Evaluation Metrics

As has been mentioned, item recommendation from implicit feedback is typically regarded as a top-N ranking problem. As a result, the evaluation is typically based on the standard ranking metrics. Despite this, there are several other ways to evaluate a recommender system, such as novelty (Hurley and Zhang, 2011), serendipity (Lu et al., 2012) and diversity (Hurley and Zhang, 2011). Since this thesis mainly focuses on improving recommendation accuracy, we only evaluate recommendation quality by accuracy metrics. In the remainder of this thesis, we will evaluate both the proposed recommendation models and baselines on popular ranking metrics, which are "translated" from the information retrieval field. They are Precision@N and Recall@N (denoted by Pre@N and Rec@N respectively), Normalized Discounted Cumulative Gain (NDCG) (McFee and Lanckriet, 2010) and Mean Reciprocal Rank (MRR) (Shi et al., 2012b), and one (binary) classification metric, i.e., Area Under ROC Curve (AUC). We perform evaluation on the testing set and compute the overlap between the predicted items and ground truth items with the following ranking metrics. The ranking metrics are defined as follows:

- Precision and recall are binary metrics used for estimating binary output. Since users care more about the items ranked at the top positions, it makes more sense to calculate precision and recall for the first N items instead of all items. N in practice can be assigned as a small value, such as 5, 10 or 100 etc. Pre@N and Rec@N are defined as follows:
$$\text{Pre@N} = \frac{tp_u}{tp_u + fp_u}, \quad \text{Rec@N} = \frac{tp_u}{tp_u + tn_u} \tag{2.22}$$
where tp_u is the number of items contained in both the ground truth and the top-N rank list predicted by algorithms; fp_u is the number of items contained in the top-N rank list but not in the ground truth; and tn_u is the number of items contained in the ground truth but not in the top-N rank list (Li et al., 2015b). The final performance reported is an average of individual metric of all users, which applies to all other metrics below.

- NDCG measures the performance of a recommender system based on the

graded relevance of the recommended items and is defined as

$$NDCG_{|I|} = \frac{DCG_{|I|}}{IDCG_{|I|}}, DCG_{|I|} = \sum_{i=1}^{|I|} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{2.23}$$

where rel_i represents the relevance score of the candidate item at the position i, here we use a binary value for quantity ($\text{rel}_i = 1$ if it is relevant, otherwise $\text{rel}_i = 0$). $IDCG_{|I|}$ is calculated from the ground truth. An intuitive example is given: assuming the ground truth of the item list is $\{1, 1, 0, 0, 0\}$ and the predicted label is $\{0, 1, 0, 0, 1\}$, where '1' and '0' denote positive and unobserved item, $IDCG_{|I|}$ is $\frac{1}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} = 1.63$, and $DCG_{|I|}$ is $\frac{1}{\log_2(2+1)} + \frac{1}{\log_2(5+1)} = 1.01$, $NDCG_{|I|}$ is $1.01/1.63 = 0.62$. Similar to Pre@N and Rec@N, we can also use NDCG@N for true evaluation, although NDCG itself is a ranking metric. This also holds for the MRR metric.

- Reciprocal Rank (Shi et al., 2012b) measures the rank of the first relevant item in the item list for a user, which is defined as

$$RR = 1/\text{rank}_i \tag{2.24}$$

The Mean Reciprocal Rank (MRR) is the average of the reciprocal ranks of all users.

$$MRR = \frac{1}{|U|} \sum_{u=1}^{|U|} 1/\text{rank}_i \tag{2.25}$$

- AUC (Rendle et al., 2009a) is widely used in classification task to evaluate which model predicts the classes best. Item recommendation can also be explained as classification task, i.e., to classify whether the positive items are ranked higher than the negative ones. AUC implies the recommender ability to distinguish positive examples from negative examples. The average AUC is usually defined as

$$\frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{|I_u||I \backslash I_u|} \sum_{i \in |I_u|} \sum_{j \in |I \backslash I_u|} \delta(\hat{y}_{uij} > 0) \tag{2.26}$$

where

$$\delta(\hat{y}_{uij} > 0) = \begin{cases} 1, & \hat{y}_{uij} > 0 \\ 0, & \text{else} \end{cases} \tag{2.27}$$

and $|I_u|$ denotes the size of observed items for user u. Note that AUC is

position-dependent by taking account of overall ranking performance.

Part II
SGD with Negative Sampling

Learning from large-scale unobserved implicit feedback is computationally very expensive in practice. To address this issue, an intuitive way is to perform negative sampling (NS), i.e., sampling a fraction of non-observe items as negative for training. In this part, we will introduce three NS-related implicit recommendation models, namely, LambdaFM (Lambda Factorization Machine) in Chapter 3, BoostFM (Boost Factorization Machines) in Chapter 4, and GeoBPR (Geographical Bayesian Personalized Ranking) in Chapter 5. For each algorithm, we conduct extensive experiments to evaluate its performance with state-of-the-art baseline algorithms.

Chapter 3

Lambda Factorization Machines

In this chapter, we introduce Lambda Factorization Machines (LambdaFM), which is state-of-the-art negative sampling based recommendation model specifically designed for implicit feedback. We use Factorization Machines (FM) as the prediction function since it is a generic model and well suited for both content/context-based recommendation and basic collaborative filtering (CF) settings. We provide the motivation from item ranking perspective and solve it by negative sampling strategy.

Previous work —Pairwise Ranking Factorization Machines (PRFM) (Qiang et al., 2013)— that learns FM by standard pairwise loss functions and uniform sampling have achieved great success. However, we argue that good recommenders particularly emphasize on the accuracy near the top of the ranked list, and typical pairwise loss functions might not match well with such a requirement since they are position-independent (Burges). We thus demonstrate, both theoretically and empirically, PRFM models usually lead to non-optimal item recommendation results due to such a mismatch. Inspired by the success of LambdaRank, we introduce Lambda Factorization Machines (LambdaFM), which is particularly intended for optimizing ranking performance for implicit feedback. We also point out that the original lambda function suffers from the issue of expensive computational complexity in such settings due to a large amount of unobserved feedback. Hence, instead of directly adopting the original lambda strategy, we create three effective negative sampling (NS) based lambda surrogates by conducting a theoretical analysis for lambda from the top-N optimization perspective. Further, we prove that the proposed lambda samplers are generic

and applicable to a large set of pairwise ranking loss functions. Extensive experiments demonstrate that LambdaFM significantly outperforms state-of-the-art algorithms on three real-world datasets in terms of four popular ranking measures.

This chapter is mainly based on our previous work "LambdaFM: Learning Optimal Ranking with Factorization Machines Using Lambda Surrogates" (Yuan et al., 2016b) published in Conference on Information and Knowledge Management (CIKM) 2016 with DOI: http://dx.doi.org/10.1145/2983323.2983758.

3.1 Introduction

Content/context-based recommendation is very prevalent in real-world recommender systems. Thus, we target at solving the problem in this setting. To leverage the content which users or items are associated with, several effective models have been proposed, among which Factorization Machines (FM) (Rendle, 2012) gain much popularity due to its elegant theory in seamless integration of sparse content and context information. As mentioned before, in the whole thesis, we focus on recommendation problems from implicit feedback. To address both content/context and implicit feedback scenarios, the ranking based FM algorithm by combining pairwise learning-to-rank (LtR) techniques (PRFM (Qiang et al., 2013)) has been investigated.

To our best knowledge, PRFM works much better than the original pointwise FM in the settings of implicit feedback with the same sampling strategy. Nevertheless, it is reasonable to argue that pairwise learning is position-independent: an incorrect pairwise-wise ordering at the bottom of the list impacts the score just as much as that at the top of the list (McFee and Lanckriet, 2010). However, for top-N item recommendation task, the learning quality is highly position-dependent: the higher accuracy at the top of the list is more important to the recommendation quality than that at the low-position items, reflected in the rank biased metrics such as Normalized Discounted Cumulative Gain (NDCG) (Pan and Chen, 2013) and Mean Reciprocal Rank (MRR) (Shi et al., 2012b). Hence, pairwise loss might still be a suboptimal scheme for ranking tasks. In this chapter, we conduct detailed analysis from the top-N optimization perspective, and shed

light on how PRFM results in non-optimal ranking in the setting of implicit feedback. Besides the theoretical analysis, we also provide the experimental results to reveal the inconsistency between the pairwise classification evaluation metric and standard ranking metrics (e.g., AUC (Rendle et al., 2009b) vs. MRR).

Inheriting the idea of differentiating pairs in LambdaRank (Quoc and Le, 2007), we propose LambdaFM, an advanced variant of PRFM by directly optimizing the rank biased metrics. We point out that the original lambda function (Quoc and Le, 2007) of LambdaRank is computationally intractable for implicit recommendation task due to a large number of unobserved feedback. To tackle such a problem, we implement three alternative surrogates (i.e., samplers) based on the analysis of the lambda function. Furthermore, we claim that the proposed lambda surrogates are more general and can be applied to a large class of pairwise loss functions. By applying these loss functions, a family of PRFM and LambdaFM algorithms have been developed. Finally, we perform thorough experiments on three real-world datasets and compare LambdaFM with state-of-the-art approaches. Our results demonstrate that LambdaFM noticeably outperform a number of baseline models in terms of four standard ranking evaluation metrics.

In the chapter, we will show that the performance difference of PRFM and LambdaFM is because of the different sampling strategies (uniform sampler vs. non-uniform sampler). Both our theoretical analysis and experiments will support our thesis statements (1) and (2).

3.2 Related Work

The approach presented in this work is rooted in research areas of content/context-based recommendation and Learning-to-Rank (LtR) techniques. Hence, we discuss the most relevant previous contribution in each of the two areas and position our work with respect to them.

3.2.1 Content/Context-based Recommender Systems

Researchers have devoted a lot of efforts to content/context-based (or feature-based) recommender systems. Early work performed pre- or post-filtering of the

input data to make standard methods content/context-aware, and thus ignoring the potential interactions between different feature variables. Recent research mostly focuses on integrating the user context (or item content) features into factorization models. Two lines of contributions have been presented to date, one based on tensor factorization (TF) (Karatzoglou et al., 2010) and the other on Factorization Machines (FM) (Rendle, 2012) as well as its variant SVDFeature (Chen et al., 2012). The type of feature by TF is usually limited to categorical variables. By contrast, the interactions of context by FM are more general, i.e., not limited to categorical ones. On the other hand, both approaches are originally designed for the rating prediction task (Cremonesi et al., 2010), which are based on the explicit user feedback. However, as mentioned before, in most real-world scenarios only implicit user behavior is observed and there is no explicit rating (Rendle and Freudenthaler, 2014; Rendle et al., 2009b). As mentioned before, the item recommendation from implicit feedback is typically formulated as a ranking problem since users care more about items that are ranked at the top position of the item list. Hence, methods by combining LtR and feature-based models (e.g., TF and FM) provide a series of solutions (Shi et al., 2014, 2012a). Detailed literature regarding learning-to-rank is discussed in the next section.

3.2.2 Learning-to-Rank

Learning-to-Rank (LtR) techniques have been attracting broad attention due to its effectiveness and importance in machine learning community. There are two major approaches in the field of LtR, namely pairwise (Burges et al., 2005; Rendle et al., 2009b; Freund et al., 2003) and listwise approaches (Shi et al., 2012b,a). In the pairwise settings, LtR problem is approximated by a classification problem, and thus existing methods in classification can be directly applied. For example, PRFM (Rendle, 2012) applies FM as the ranking function to model the pairwise interactions of features, and optimizes FM by maximizing the AUC metric. However, it has been pointed out in previous Information Retrieval (IR) literature that pairwise approaches are designed to minimize the classification errors of objective pairs, rather than errors in ranking of items (Liu et al., 2009). In other words, the pairwise loss does not inversely correlate with the ranking measures such as Normalized Discounted Cumulative Gain (NDCG) and MAP. By

contrast, listwise approaches solve this problem in a more elegant way where the models are formalized to directly optimize a specific list-level ranking metric. For example, TFMAP (Shi et al., 2012a) optimizes a TF model by maximizing the Mean Average Precision (MAP) metric; similar work by CARS2 (Shi et al., 2014) studied to use the same listwise loss functions to learn a novel TF model; However, an obvious drawback of the listwise method is that it is generally non-trivial to directly optimize the ranking performance measures because they are either non-continuous or indifferentiable, e.g., NDCG, MAP and MRR. As a result, the listwise loss functions have to rely on the design of approximations or bounds of a differentiable loss surrogate, which easily leads to sub-optimal results (Shi et al., 2014). The other way is to add listwise information into pairwise learning. The most typical work is LambdaRank (Quoc and Le, 2007), where the change of NDCG of the ranking list if switching the item pair is incorporated into the pairwise loss in order to reshape the model by emphasizing its learning over the item pairs leading to large NCDG drop.

It is worth noticing that previous LtR models (e.g., Ranking SVM (Herbrich et al., 1999), RankBoost (Freund et al., 2003), RankNet (Burges et al., 2005), ListNet (Cao et al., 2007), LambdaRank (Quoc and Le, 2007)) were originally proposed for IR tasks with dense features, which might not be directly applied in recommender systems (RS) with huge sparse context feature space. Besides, RS target at personalization, which means each user should attain one set of parameters for personalized ranking, whereas the conventional LtR normally learns one set of global parameters (Cao et al., 2007), i.e., non-personalization. Hence, in this work we build our contributions on the state-of-the-art algorithm of PRFM. By a detailed analysis for the ranking performance of PRFM, we present LambdaFM motivated by the idea of LambdaRank. However, computing such a lambda poses an efficiency challenge in learning the model. By analyzing the function of lambda, we present three alternative surrogate strategies that are capable of achieving equivalent performance.

3.3 Preliminaries

In this section, we first recapitulate the idea and implementation of PRFM (Qiang et al., 2013) based on the pairwise cross entropy loss and uniform negative sampling. Then we show that PRFM suffers from a suboptimal ranking for top-N item recommendations. Motivated by this, we devise the LambdaFM algorithm by applying the idea from LambdaRank.

3.3.1 Pairwise Ranking Factorization Machines

In the context of recommendation, let C be the whole set of contexts (including user and user-related features, aka user-field) and I the whole set of items (including item and item-related features, aka item-field). Assume that the learning algorithm is given a set of pairs of items $(i, j) \in I$ for context $c \in C$, together with the desired target value $\overline{P}_{ij}^c$ for the posterior probability, and let $\overline{P}_{ij}^c \in \{0, 0.5, 1\}$ be defined as 1 if i is more relevant than j given c, 0 if item i is less relevant, and 0.5 if they have the same relevance. The cross entropy (CE) (Burges et al., 2005) loss is defined as

$$L = \sum_{c \in C} \sum_{i \in I} \sum_{j \in I} -\overline{P}_{ij}^c \log P_{ij}^c - \left(1 - \overline{P}_{ij}^c\right) \log \left(1 - P_{ij}^c\right) \tag{3.1}$$

where P_{ij}^c is the modeled probability

$$P_{ij}^c = \frac{1}{1 + \exp(-\sigma\left(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j)\right))} \tag{3.2}$$

where σ determines the shape of sigmoid with 1 as default value, $\mathbf{x} \in \mathbb{R}^n$ denotes the input vector, and $\hat{y}(\mathbf{x})$ is the ranking score computed by 2-order FM (i.e., Eq. 2.9 in Chapter 2):

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{d=1}^{n} w_d x_d + \frac{1}{2} \sum_{f=1}^{k} ((\sum_{d=1}^{n} v_{d,f} x_d)^2 - \sum_{d=1}^{n} v_{d,f}^2 x_d^2) \tag{3.3}$$

where n is the number of context variables, k is a hyper-parameter that denotes the dimensionality of latent factors, and w_0, w_d, $v_{d,f}$ are the model parameters to be estimated, i.e., $\Theta = \{w_0, w_1, ..., w_n, v_{1,1}, ..., v_{n,k}\} = \{w_0, \mathbf{w}, \boldsymbol{V}\}$. As previously mentioned, we are focusing on the implicit recommendation problem. To simplify Eq. (3.1), we formalize implicit feedback training data for context c as

$$D_c = \{\langle i, j \rangle_c | i \in I_c \wedge j \in I \backslash I_c\} \tag{3.4}$$

where I_c represents the set of positive context-item pairs, i and j are an observed and unobserved item for c, respectively. Thus, we have $\overline{P}_{ij}^{c}$=1. Now the CE loss function and gradient for each $\langle i,j\rangle_c$ pair in our settings becomes

$$L(\langle i,j\rangle_c) = \log(1+\exp(-\sigma(\hat{y}(\mathbf{x}^i)-\hat{y}(\mathbf{x}^j))) \tag{3.5}$$

$$\frac{\partial L(\langle i,j\rangle_c)}{\partial\theta} = \lambda_{i,j}(\frac{\partial\hat{y}(\mathbf{x}^i)}{\partial\theta} - \frac{\partial\hat{y}(\mathbf{x}^j)}{\partial\theta}) \tag{3.6}$$

where $\lambda_{i,j}$ is the learning weight[1] for $\langle i,j\rangle_c$ pair and is defined as

$$\lambda_{i,j} = \frac{\partial L(\langle i,j\rangle_c)}{\partial(\hat{y}(\mathbf{x}^i)-\hat{y}(\mathbf{x}^j))} = -\frac{\sigma}{1+\exp(\sigma(\hat{y}(\mathbf{x}^i)-\hat{y}(\mathbf{x}^j)))} \tag{3.7}$$

According to the property of Multilinearity (Rendle, 2012), the gradient of Eq. (3.3) can be derived as

$$\frac{\partial\hat{y}(\mathbf{x}^i)}{\partial\theta} = \begin{cases} x_d^i & \text{if } \theta \text{ is } w_d \\ x_d^i\sum_{l=1}^{n} v_{l,f}x_l^i - v_{d,f}{x_d^i}^2 & \text{if } \theta \text{ is } v_{d,f} \end{cases} \tag{3.8}$$

By combining Eqs. (3.5)-(3.8), we obtain

$$\begin{aligned} w_d &\leftarrow w_d - \eta(\lambda_{i,j}(x_d^i - x_d^j) + \gamma_{w_d}w_d) \\ v_{d,f} &\leftarrow v_{d,f} - \eta(\lambda_{i,j}(\sum_{l=1}^{n} v_{l,f}(x_d^i x_l^i - x_d^j x_l^j) - v_{d,f}({x_d^i}^2 - {x_d^j}^2)) + \gamma_{v_{d,f}}v_{d,f}) \end{aligned} \tag{3.9}$$

where γ_θ (i.e., γ_{w_d}, $\gamma_{v_{d,f}}$) is a hyper-parameter for the L2 regularization. To handle the large number of unobserved feedback (i.e., $I\backslash I_c$), the common practice is to apply Stochastic Gradient Descent (SGD) with negative sampling (Rendle et al., 2009b). Finally, the pseudo algorithm of PRFM in the settings of implicit feedback is shown in Algorithm 1, where Line 7 denotes negative sampling with the uniform item distribution.

3.3.2 Lambda Motivation

Top-N item recommendation is often referred to as a ranking task, where ranking measures like Normalized Discounted Cumulative Gain (NDCG) are widely adopted to evaluate the recommendation accuracy. However, it has been pointed out in previous IR literature (e.g., Quoc and Le (2007)) that pairwise loss functions might not match well with these measures. For easy reading, we start to state the problem by an intuitive example in the form of implicit feedback.

[1] $\lambda_{i,j}$ can be read as how much influence $\langle i,j\rangle_c$ has for updating Θ.

Algorithm 1 Ranking FM Learning

Input: Training dataset, regularization parameters γ_θ, learning rate η
Output: Parameters $\Theta = (\mathbf{w}, \boldsymbol{V})$
Initialize Θ: $\mathbf{w} \leftarrow (0, ..., 0)$; $\boldsymbol{V} \sim \mathcal{N}(0, 0.1)$;
repeat
    Uniformly draw c from C ;
    Uniformly draw i from I_c ;
    Uniformly draw j from $I \backslash I_c$;
    for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
        Update w_d
    end for
    for $f \in \{1, ..., k\}$ **do**
        for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
            Update $v_{d,f}$
        end for
    end for
until convergence
return Θ

Figure 3.1 is a schematic that illustrates the relations of pairwise loss and NDCG. By comparison of (b) and (c), we observe two mismatches between them. First, the pairwise errors decrease from eight (b) to seven (c), along with the value of NDCG decreasing from 0.790 to 0.511. However, an ideal recommender model is supposed to increase the NDCG value with the drop of pairwise errors. Second, in (c), the arrows denote the next optimization direction and strength, and thus can be regarded as the gradients of PRFM. Although both arrows reduce the same amount of pairwise errors (i.e., three), the red dashed arrows lead to more gains as desired. This is because the new NDCGs, after the optimization, are 0.624 and 0.832 by the black solid and red dashed arrows, respectively.

According to the above counterexamples, we clearly see that the pairwise loss function used in the optimization process is not a desired one as reducing its value does not necessarily increase the ranking performance. This means, PRFM might still be a suboptimal scheme for top-N recommendations. This motivates us to study whether PRFM can be improved by concentrating on maximizing the ranking measures.

To overcome the above challenge, lambda-based methods (Quoc and Le, 2007) in the field of IR have been proposed by directly optimizing the ranking biased

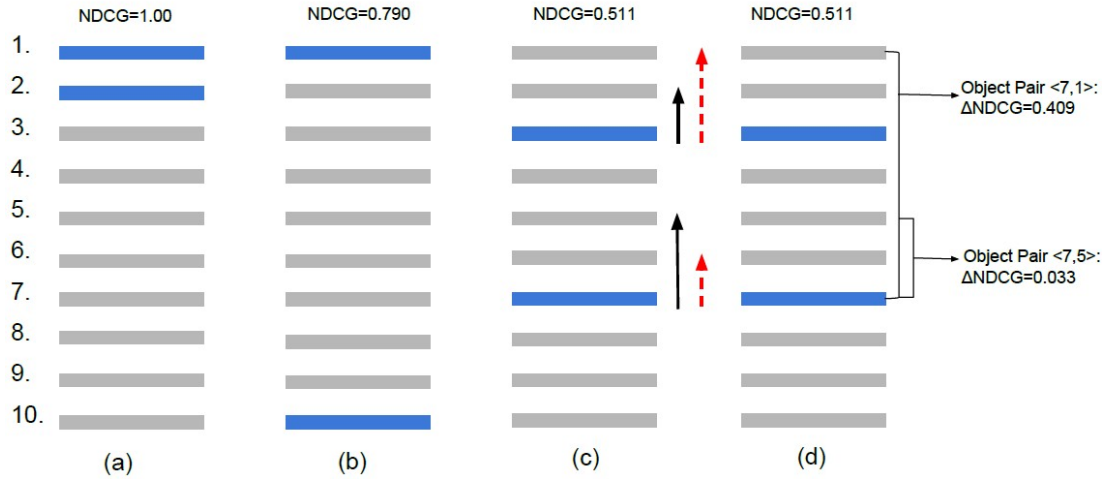


Figure 3.1: A set of items ordered under a given context (e.g., a user) using a binary relevance measure. The blue bars represent relevant items for the context, while the light gray bars are those not relevant. (a) is the ideal ranking; (b) is a ranking with eight pairwise errors; (c) and (d) are a ranking with seven pairwise errors by moving the top items of (b) down two rank levels, and the bottom preferred items up three. The two arrows (black solid and red dashed) of (c) denote two ways to minimize the pairwise errors. (d) shows the change in NDCG by swapping the orders of two items (e.g., item 7 and item 1).

performance measures. Inheriting the idea of LambdaRank, we design a new recommendation model named Lambda Factorization Machines (LambdaFM), an extended version of PRFM for solving top-N context-based recommendations in implicit feedback domains. Following this idea, we can design a similar lambda function as $f(\lambda_{i,j}, \zeta_c)$, where ζ_c is the current item ranking list for context c. With NDCG as target, $f(\lambda_{i,j}, \zeta_c)$ is given

$$f(\lambda_{i,j}, \zeta_c) = \lambda_{i,j} |\triangle NDCG_{ij}| \tag{3.10}$$

where $\triangle NDCG_{ij}$ is the NDCG difference of the ranking list under a given context if the positions (ranks) of items i, j get switched. LambdaFM can be implemented by replacing $\lambda_{i,j}$ with $f(\lambda_{i,j}, \zeta_c)$ in Eqs. (3.9).

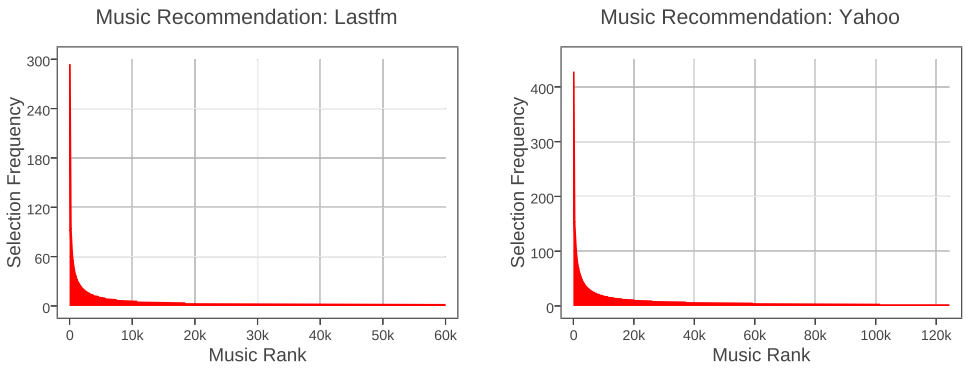
We find that the above implementation is reasonable for multi-label learning tasks in typical IR tasks, but not applicable into implicit feedback settings. This is because to calculate $\triangle NDCG_{ij}$ it requires to compute scores of all items to obtain the rankings of item i and j. For the traditional LtR, the candidate URLs returned for a given query in training datasets have usually been limited

to a small size. However, for recommendation, since there is no query at all, all unobserved (non-positive) items should be regarded as candidates, which leads to a very large size of candidates (Zhang et al., 2013). Thus the complexity to calculate each training pair in implicit feedback settings becomes $O(|I| \cdot T_{pred})$, where T_{pred} is the time for predicting a score by Eq. (3.3). Note that since the NDCG value is determined by the parameters of FM which however are updated in each SGD step. This means the whole ranking orders are totally changed after each SGD update. In other words, it is also meaningless to calculate the NDCG value in advance. That is, the original lambda function devised for LambdaRank is impractical for our settings.

3.4 Lambda Strategies

To handle the above problem, we need to revisit Eqs. (3.10), from which $\triangle NDCG_{ij}$ can be interpreted as a weight function that it rewards a training pair by raising the learning weight (i.e., $\lambda_{i,j}$) if the difference of NDCG is larger after swapping the two items, otherwise it penalizes the training pair by shrinking the learning weight. Suppose an ideal lambda function $f(\lambda_{i,j}, \zeta_c)$ for each training pair $\langle i, j \rangle_c$ is given with the current item ranking list ζ_c, if we design a scheme that generates the training item pair $\langle i, j \rangle_c$ with the probability proportional to $f(\lambda_{i,j}, \zeta_c)/\lambda_{i,j}$ (just like $\triangle NDCG_{ij}$ in Eqs. (3.10)), then we are able to construct an almost equivalent training model. In other words, a higher proportion of item pairs should be drawn according to the probability distribution $p_j \propto f(\lambda_{i,j}, \zeta_c)/\lambda_{i,j}$ if they have a larger $\triangle NDCG$ by swapping.

On the other hand, we observe that item pairs with a larger $\triangle NDCG$ contribute more to the desired loss and lead to a larger NDCG value. In the following, we refer to a training pair $\langle i, j \rangle_c$ as an informative item pair, if $\triangle NDCG_{ij}$ is larger after swapping i and j than another pair $\langle i, j' \rangle$. The unobserved item j is called a good or informative item. This leads to two research questions arise: (1) how to judge which item pairs are more informative? and (2) how to pick up informative items? Figure 3.1 (d) gives an example of which item pairs should have a higher weight. Obviously, we observe that the quantity of $\triangle NDCG_{71}$ is larger than that of $\triangle NDCG_{75}$. This indicates $\triangle NDCG_{ij}$ is likely to be larger



(a) Lastfm (b) Yahoo

Figure 3.2: Item popularity on the Lastfm (short for Last.fm) and Yahoo datasets plotted in the linear scale.

if unobserved (e.g., unselected) items have a higher rank (or a relatively higher score by Eq. (3.3)). This is intuitively correct as the high ranked non-positive items hurt the ranking performance (users' feelings) more than the low ranked ones. The other side of the intuition is that low ranked positive items contribute similarly as high ranked non-positive items during the training process, which would be provided with more details later.

In the following, we introduce three intuitive lambda-motivated sampling methods (i.e., lambda surrogates) for addressing the two research questions and refer to PRFM with suggested lambda surrogates as LambdaFM (LFM for short in some places). The above discussions and the following solutions are consistent with our **statement (1)** and **statement (2)**.

3.4.1 Static and Context-independent Sampler

We believe that a popular item $j \in I \backslash I_c$ is supposed to be a reasonable substitute for a high ranked non-positive item, i.e., the so-called informative item. The reason is intuitive as an ideal learning algorithm is expected to restore the pairwise ordering relations and rank most positive items with larger ranking scores than non-positive ones. As we know, the more popular an item is, the more times it acts as a positive one. Besides, popular items are more likely to be suggested by a recommender in general, and thus they have higher chances to be positive items. That is, the observation that a context is more relevant to an item over

a popular one provides more information to capture the user's potential interest than that he/she prefers an item over a very unpopular one. Thus, it is reasonable to assign a higher weight to non-positive items with high popularity, or sampling such items with a higher probability.

In fact, it has been recognized that item popularity distribution in most real-world recommendation datasets has a heavy tail (Park and Tuzhilin, 2008; Lu et al., 2012) (also see Figure 3.2), following an approximate power-law or exponential distribution. Accordingly, most non-positive items drawn by uniform sampling in Algorithm 1 are unpopular due to the long tail distribution, and thus contribute less to the desired loss function. Based on the analysis, it is reasonable to present a popularity-aware sampler to replace the uniform one before performing each SGD. Let p_j denote the sampling distribution for item j. In this work, we draw unobserved items with the probability proportional to the empirical popularity distribution, e.g., an exponential distribution.

$$p_j \propto \exp(-\frac{r(j)+1}{|I| \times \rho}), \rho \in (0, 1] \tag{3.11}$$

where $r(j)$ represents the rank of item j among all items I according to the overall popularity, ρ is the parameter of the distribution. Therefore, Line 6 in Algorithm 1 can be directly replaced as the above sampler, i.e., Eq. (3.11). Hereafter, we denote PRFM with the static popularity-aware sampler as LFM-S. Interestingly, in Rendle and Freudenthaler (2014), by analyzing from the gradient vanishing perspective, the authors found that the popularity-based sampler hurts the recommendation accuracy although it converges much faster. While in our evaluation we empirically found that BPR with the popularity sampler can significantly improve the basic uniform sampler (Yuan et al., 2016b). The reason may be because in Rendle and Freudenthaler (2014) they oversampled only a small portion of very popular unobserved items based on the empirical distribution while most relatively popular items are missing.

The proposed sampler has three important properties:

- *Static.* The sampling procedure is static, i.e., the distribution does not change during the training process.
- *Context-Invariant.* The item popularity is independent of context information according to Eq. (3.11).

Algorithm 2 Rank-Aware Dynamic Sampling

1: **Require:** Unobserved item set $I \backslash I_c$, scoring function $\hat{y}(\cdot)$, parameter m, ρ
2: Draw sample $j_1,...,j_m$ uniformly from $I \backslash I_c$
3: Compute $\hat{y}(\boldsymbol{x}^{j_1}),...,\hat{y}(\boldsymbol{x}^{j_m})$
4: Sort $j_1,...,j_m$ by descending order, i.e., $r(j_m) \propto \frac{1}{\hat{y}(\boldsymbol{x}^{j_m})}$
5: Return one item from the sorted list with the exponential distribution $p_j \propto \exp(-(r(j)+1)/(m \times \rho))$.

- *Efficient.* The sampling strategy does not increase the computational complexity since the popularity distribution is static and thus can be calculated in advance.

3.4.2 Dynamic and Context-aware Sampler

The second sampler is a dynamic one which is able to change the sampling procedure while the model parameters Θ are updated. The main difference of dynamic sampling is that the item rank is computed by their scores instead of global popularity. As it is computationally expensive to calculate all item scores in the item list given a context, which is different from the static item popularity calculated in advance, we first perform a uniform sampling to obtain m candidates. Then we calculate the candidate scores and sample the candidates also by an exponential distribution. As the first sampling is uniform, the sampling probability density for each item is equivalent with that from the original (costy) global sampling. A straightforward algorithm can be implemented by Algorithm 2. We denote PRFM with the dynamic sampler as LFM-D. It can be seen this sampling procedure is dynamic and context-aware.

- The sampler selects non-positive items dynamically according to the current item ranks which are likely to be different at each update.
- The item rank is computed by the FM function (i.e., Eq. (3.3)), which is clearly context-aware.
- The complexity for parameter update of each training pair is $O(mT_{\text{pred}}$ (Line 3)$+m \log m$ (Line 4)), where the number of sampling items m can be set to a small value (e.g., m=10, 20, 50). Thus introducing the dynamic sampler will not increase the time complexity much.

3.4.3 Rank-aware Weighted Approximation

The above two surrogates are essentially based on non-positive item sampling techniques, the core idea is to push non-positive items with higher ranks down from the top positions. Based on the same intuition, an equivalent way is to pull positive items with lower ranks up from the bottom positions. That is, we have to place less emphasis on the highly ranked positives and more emphasis on the lowly ranked ones. Specifically, if a positive item is ranked top in the list, then we use a reweighting term $\Gamma\big(r(i)\big)$ to assign a smaller weight to the learning weight $\lambda_{i,j}$ such that it will not cost the loss too much. However, if a positive item is not ranked at top positions, $\Gamma\big(r(i)\big)$ will assign a larger weight to the gradient, pushing the positive item to the top.

$$f(\lambda_{i,j}, \zeta_u) = \Gamma\big(r(i)\big)\lambda_{i,j} \tag{3.12}$$

By considering maximizing objectives e.g., reciprocal rank, we define $\Gamma\big(r(i)\big)$ as

$$\Gamma\big(r(i)\big) = \frac{1}{\Gamma\big(I\big)} \sum_{r=0}^{r(i)} \frac{1}{r+1} \tag{3.13}$$

where $\Gamma\big(I\big)$ is the normalization term calculated by the theoretically lowest position of a positive item, i.e.,

$$\Gamma\big(I\big) = \sum_{r \in I} \frac{1}{r+1} \tag{3.14}$$

However, the same issue occurs again, i.e., the rank of the positive item i is unknown without computing the scores of all items. In contrast with non-positive item sampling methods, sampling positives is clearly useless as there are only a small fraction of positive items (i.e., $|I_c|$), and thus all of them should be utilized to alleviate the sparsity. Interestingly, we observe that it is possible to compute the approximate rank of positive item i by sampling one incorrectly-ranked item. More specifically, given a $\langle c, i \rangle$ pair, we repeatedly draw an item from I until we obtain an incorrectly-ranked item j such that $\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) \leq \varepsilon$ and $\overline{P}^c_{ij}$=1, where ε is a positive margin value[1]. Let T denote the size of sampling trials before obtaining such an item. Apparently, the sampling process corresponds to

[1] We hypothesize that item j is ranked higher than i for context c only if $\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) \leq \varepsilon$, the default value of ε is set to 1.

a geometric distribution with parameter $p = \frac{r(i)}{|I|-1}$. Since the number of sampling trials can be regarded as the expectation of parameter p, we have $T \approx \lceil \frac{1}{p} \rceil = \lceil \frac{|I|-1}{r(i)} \rceil$, where $\lceil \cdot \rceil$ is the ceiling function[1]. Finally, by using the estimation, we rewrite the gradient

$$\frac{\partial L(\langle i,j \rangle_u)}{\partial \theta} = \frac{\sum_{r=0}^{\lceil \frac{|I|-1}{T} \rceil} \frac{1}{r+1}}{\Gamma(I)} \lambda_{i,j} \left(\frac{\partial \hat{y}(\mathbf{x}^i)}{\partial \theta} - \frac{\partial \hat{y}(\mathbf{x}^j)}{\partial \theta} \right) \tag{3.15}$$

We denote the proposed algorithm based weighted approximation as LFM-W in Algorithm 3. In this algorithm, we iterate through all positive items $i \in I_c$ for each context and update the model parameters $\mathbf{w}$, $\boldsymbol{V}$ until the procedure converges. In each iteration, given a context-item pair, the sampling process is first performed so as to estimate violating condition and obtain a desired item j. Note that item j is not limited to a non-positive item, since it can also be drawn from the positive collection I_c with a lower preference than i. Once j is chosen, we update the model parameters by applying the SGD method.

We are interested in investigating whether the weight approximation yields the right contribution for the desired ranking loss. Based on Eq. (3.15), it is easy to achieve a potential loss induced by the violation of $\langle i,j \rangle_c$ pair using back-stepping approach

$$L(\langle i,j \rangle_c) = \frac{\sum_{r=0}^{\lceil \frac{|I|-1}{T} \rceil} \frac{1}{r+1}}{\Gamma(I)} \log(1 + \exp(-\sigma(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j)))) \tag{3.16}$$

To simplify the function, we first omit the denominator term since it is a constant value, then we replace the CE loss with the 0/1 loss function.

$$\frac{\partial L(\langle i,j \rangle_c)}{\partial \theta} = \sum_{r=0}^{r(i)} \frac{1}{r+1} I\left[\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j)\right] \tag{3.17}$$

where $I[\cdot]$ is the indicator function, $I[h] = 1$ when h is true, and 0 otherwise. Thus, we have $I\left[\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j\right] = 1$ in all the cases because r is smaller than $r(i)$. Moreover, for each positive item i, there are $r(i)$ items that are ranked higher than i (i.e., item j), and thus each j has the probability of $p = \frac{1}{r(i)}$ to be picked.

[1] The idea here is inspired by Weston et al. (2011), which was designed to solve the image classification problem.

Finally, the formulation of the loss for each context c is simplified as[1]

$$L_c = \sum_{i \in I_c} \sum_{r=0}^{r(i)} \frac{1}{r+1} \tag{3.18}$$

As previously demonstrated in Section 3.3.2, the difference of pairwise losses between (b) and (c) in Figure 3.1 is inconsistent with that of NDCG values. Instead of using standard pairwise loss function, here we also give an intuitive example to show the merit of our proposed method in reducing loss and improving the accuracy. We compute the losses based on Eq. (3.18), from which we achieve the losses of (b) and (c) are 2.93 and 4.43, respectively. This means, a smaller loss leads to a larger NDCG value, and vice versa. Similarly, we notice that the black (solid) arrows in (c) after movement lead to a larger loss than the red (dashed) ones, and in reason, the new NDCG value generated by the black ones is smaller than that by the red ones. Therefore, the movement direction and strength of red arrows is the right way to minimize a larger loss, which also demonstrates the correctness of the proposed lambda strategy. Based on Eq. (3.18), one can draw similar conclusions using other examples.

Regarding the properties of LFM-W, we observe that the weight approximation procedure is both dynamic (Line 9-13 of Algorithm 3) and context-aware (Line 11) during each update, similarly like LFM-D. Moreover, LFM-W is able to leverage both binary and graded relevance datasets (e.g. URL click numbers, POI check-ins), whereas LFM-S and LFM-D cannot learn the count information from positives. In terms of the computational complexity, using the gradient calculation in Eq. (3.15) can achieve important speedups. The complexity to compute $\triangle NDCG$ is $O(T_{\text{pred}}|I|)$ while the complexity with weighted approximation becomes $O(T_{\text{pred}}T)$. Generally, we have $T \ll |I|$ at the start of the training and $T < |I|$ when the training is stable. The reason is that at the beginning, LambdaFM is not well trained and thus it is quick to find an offending item j, which needs only a very small T, i.e., $T \ll |I|$, when the training converges to a stable state, most positive items are already be ranked correctly, which is also our expectation, and thus T becomes larger. However, it is unlikely that all positive items are ranked correctly, so we still have $T < |I|$.

[1] Please note that L_c is non-continuous and indifferentiable, which means it is hard to be directly optimized with this form.

Algorithm 3 LFM-W Learning

Input: Training dataset, regularization parameters γ, learning rate η
Output: Parameters $\Theta = (\mathbf{w}, \boldsymbol{V})$
Initialize Θ: $\mathbf{w} \leftarrow (0, ..., 0)$; $\boldsymbol{V} \sim \mathcal{N}(0, 0.1)$;
repeat
    Uniformly draw c from C
    Uniformly draw i from I_c
    Calculate $\hat{y}(\mathbf{x}^i)$
    Set T=0
    do
        Uniformly draw j from I
        Calculate $\hat{y}(\mathbf{x}^j)$
        T+=1
    while $\big(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) > \varepsilon \parallel \overline{P}^c_{ij} \neq 1\big) \wedge \big(T < |I| - 1\ \big)$
    if $\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) \leq \varepsilon \wedge \overline{P}^c_{ij} = 1$ **then**
        Calculate $\lambda_{i,j}, \Gamma\big(I\big)$ according to Eqs. (3.7) and (3.14)
        for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
            Update w_d:
            $w_d \leftarrow w_d - \eta(\lambda_{i,j}(x^i_d - x^j_d)\frac{\sum_{r=0}^{\lceil \frac{|I|-1}{T} \rceil} \frac{1}{r+1}}{\Gamma(I)} - \gamma_{w_d} w_d)$
        end for
        for $f \in \{1, ..., k\}$ **do**
            for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
                Update $v_{d,f}$:
                $v_{d,f} \leftarrow v_{d,f} - \eta(\lambda_{i,j}(\sum_{l=1}^{n} v_{l,f}(x^i_d x^i_l - x^j_d x^j_l)$
                $-v_{d,f}({x^i_d}^2 - {x^j_d}^2))\frac{\sum_{r=0}^{\lceil \frac{|I|-1}{T} \rceil} \frac{1}{r+1}}{\Gamma\big(I\big)} - \gamma_{v_{d,f}} v_{d,f})$
            end for
        end for
    end if
until convergence
return Θ

3.5 Lambda with Alternative Losses

From Section 3.3.1, one may observe that the original lambda function was a specific proposal for the CE loss function used in RankNet (Burges et al., 2005). There exists a large class of pairwise loss functions in IR literature. The well-known pairwise loss functions, for example, can be margin ranking criterion, fidelity loss, exponential loss, and modified Huber loss, which are used in Ranking SVM (Herbrich et al., 1999), Frank (Tsai et al., 2007), RankBoost (Freund et al., 2003), quadratically smoothed SVM (Zhang, 2004), respectively. Motivated by the design ideas of these famous algorithms, we build a family of LambdaFM variants based on these loss functions and verify the generic properties of our lambda surrogates.

Margin ranking criterion (MRC):

$$L = \sum_{c \in C} \sum_{i \in I_c} \sum_{j \in I \setminus I_c} \max(0, 1 - (\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))) \tag{3.19}$$

MRC (aka, Hinge loss) assigns each positive-negative (non-positive) item pair a cost if the score of non-positive item is larger or within a margin of 1 from the positive score. Optimizing this loss is equivalent to maximizing the area under the ROC curve (AUC). Since MRC is non-differentiable, we optimize its subgradient as follows

$$\lambda_{i,j} = \begin{cases} -1 & \text{if } \hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) < 1 \\ 0 & \text{if } \hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j) \geq 1 \end{cases} \tag{3.20}$$

As is known, the pairwise violations are independent of their positions in the ranking list (Hong et al., 2013). For this reason, MRC might not optimize top-N very accurately.

Fidelity loss (FL): This loss is introduced by Tsai et al. (2007) and has been applied in Information Retrieval (IR) task and yielded superior performance. The original function regarding the loss of pairs is defined as

$$L = \sum_{c \in C} \sum_{i \in I} \sum_{j \in I} (1 - \sqrt{\overline{P}_{ij} \cdot P_{i,j}} - \sqrt{(1 - \overline{P}_{ij}) \cdot (1 - P_{i,j})}) \tag{3.21}$$

where $\overline{P}_{ij}$ and P_{ij} share the same meanings with the CE loss in Eq. (3.1). According to the antisymmetry of pairwise ordering scheme (Rendle et al., 2009b),

we simplify the FL in implicit feedback settings as follows

$$L = \sum_{c \in C} \sum_{i \in I_c} \sum_{j \in I \setminus I_c} (1 - \frac{1}{\sqrt{1 + \exp(-\hat{y}(\mathbf{x}^i) + \hat{y}(\mathbf{x}^j))}}) \tag{3.22}$$

In contrast with other losses (e.g., the CE and exponential losses), the FL of a pair is bounded between 0 and 1, which means the model trained by it is more robust against the influence of hard pairs. However, we argue that (1) FL is non-convex, which makes it difficult to optimize; (2) adding bound may cause insufficient penalties of informative pairs.

Modified Huber loss (MHL):

$$L = \sum_{c \in C} \sum_{i \in I_c} \sum_{j \in I \setminus I_c} \frac{1}{2\gamma} \max(0, 1 - (\hat{y}(\boldsymbol{x}^i) - \hat{y}(\boldsymbol{x}^j)))^2 \tag{3.23}$$

where γ is a positive constant and is set to 2 for evaluation. MHL is quadratically smoothed variant of MRC and proposed for linear prediction problems by Zhang (2004).

It is worth noting that both Fidelity and Huber loss functions have not been investigated yet in the context of recommendation. Besides, to the best of our knowledge, PRFM built on these two loss functions are novel. On the other hand, we find that exponential loss function is aggressive and seriously biased by hard pairs, which usually result in worse performance during the experiment. Thus the trial of PRFM with exponential loss have been omitted.

3.6 Experiments

We conduct experiments on the several real-world datasets to verify the effectiveness of LambdaFM (i.e., the three lambda-based negative samplers) in various settings.

3.6.1 Experimental Setup

3.6.1.1 Datasets

We use three publicly accessible Collaborative Filtering (CF) datasets for our experiments: Yelp[1] (context-venue pairs), Lastfm[2] (context-music-artist triples)

[1] https://www.yelp.co.uk/dataset_challenge
[2] http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html

Table 3.1: Basic statistics of datasets. Each entry indicates a context-item pair

Dataset	Context	Items	Artists	Albums	Entries
Yelp	10827	23115	-	-	333338
Lastfm	983	60000	25147	-	246853
Yahoo	2450	124346	9040	19851	911466

Table 3.2: Performance comparison on NDCG, MRR and AUC. For each measure, the best result is indicated in bold.

Dataset	Metrics	MP	FM	BPR	PRFM	LFM-S	LFM-D	LFM-W
Yelp	NDCG	0.1802	0.2130	0.2186	0.2186	0.2218	**0.2232**	0.2191
	MRR	0.0451	0.0718	0.0860	0.0852	0.0950	**0.0997**	0.0977
	AUC	0.8323	0.8981	0.9043	**0.9044**	0.8876	0.8787	0.874
Lastfm	NDCG	0.3452	0.3832	0.3830	0.3944	0.4095	0.4175	**0.4191**
	MRR	0.2051	0.2182	0.2588	0.2856	0.3433	0.3514	**0.3914**
	AUC	0.8506	0.9161	0.9055	**0.9209**	0.9145	0.9075	0.8949
Yahoo	NDCG	0.3109	0.3682	0.3478	0.3720	0.3791	0.3993	**0.4016**
	MRR	0.1252	0.1942	0.1909	0.2211	0.2467	0.2857	**0.3004**
	AUC	0.8425	0.9313	0.8720	**0.9357**	0.9256	0.9340	0.9273

and Yahoo music[1] (context-music-artist-album tuples). To speed up the experiments, we randomly sampling a subset of context from the Yahoo datasets, and a subset of items from the item pool of the Lastfm dataset. On the Yelp dataset, we extract data from Phoenix and follow the common practice (Rendle et al., 2009b) to filter out contexts with less than 10 interactions. The reason is because the original Yelp dataset is much sparser than Lastfm and Yahoo datasets[2], which makes it difficult to evaluate recommendation algorithms (e.g., over half users have only one entry.). The statistics of the datasets after preprocessing are summarized in Table 4.1.

3.6.1.2 Evaluation Metrics

To illustrate the recommendation quality of LambdaFM, we adopt four standard ranking metrics: Precision@N and Recall@N (denoted by Pre@N and Rec@N respectively), Normalized Discounted Cumulative Gain (NDCG) (Pan and Chen, 2013; McFee and Lanckriet, 2010) and Mean Reciprocal Rank (MRR) (Shi et al.,

[1] webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=2

[2] Cold users of both datasets have been trimmed by official provider.

2012b), and one (binary) classification metric, i.e., Area Under ROC Curve (AUC). Note that AUC is position-dependent by taking account of overall ranking performance. For each evaluation metric, we first calculate the performance of each context from the testing set, and then obtain the average performance over all contexts. More details about these metrics can be found in Chapter 2. The formulations of the evaluation metrics are given in Chapter 2.

3.6.1.3 Baseline Methods

We compare LambdaFM with four powerful baseline methods.

- **Most Popular (MP)**: It returns the context (i.e., user in this chapter) with the top-N most popular items. MP is a commonly used benchmark for the item recommendation task from implicit feedback.
- **Factorization Machines (FM)** (Rendle, 2012): FM is designed for the rating prediction task. We adapt FM for the item recommendation task by binarizing the rating value. Note that, to conduct a fair comparison, we also use the uniform sampling to make use of non-positive items.
- **Bayesian Personalized Ranking (BPR)** (Rendle et al., 2009b): It is a strong context-free recommendation algorithms specifically designed for top-N item recommendations based on implicit feedback.
- **Pairwise Ranking Factorization Machines (PRFM)** (Qiang et al., 2013): PRFM is a state-of-the-art context-based ranking algorithm, optimized to maximize the AUC metric. In this chapter, we develop several PRFM[1] algorithms by applying various pairwise loss functions introduced in Section 3.5.

3.6.1.4 Hyper-parameter Settings

To perform stochastic gradient descent (SGD), there are several critical hyper-parameters.

- **Latent dimension** k: The effect of latent factors has been well studied by previous work, e.g., Rendle et al. (2009b). For comparison purposes, the

[1] PRFM is short for PRFM with the CE loss if not explicitly declared.

approaches, e.g., Pan and Chen (2013), are usually to assign a fixed k value (e.g., $k = 30$ in our experiments) for all methods based on factorization models.

- **Regularization** γ_θ: LambdaFM has several regularization parameters, including γ_{w_d}, $\gamma_{v_{d,f}}$, which represent the regularization parameters of latent factor vectors of w_d, $v_{d,f}$, respectively. We borrow the idea from Rendle (2012) by grouping them for each factor layer, i.e., $\gamma_\pi = \gamma_{w_d}$, $\gamma_\xi = \gamma_{v_{d,f}}$. We run LambdaFM with $\gamma_\pi, \gamma_\xi \in \{0.5, 0.1, 0.05, 0.01, 0.005\}$ to find the best performance parameters.

- **Distribution coefficient** ρ: $\rho \in (0, 1]$ is specific for LFM-S and LFM-D, which is usually tuned according to the data distribution.

3.6.2 Performance Evaluation

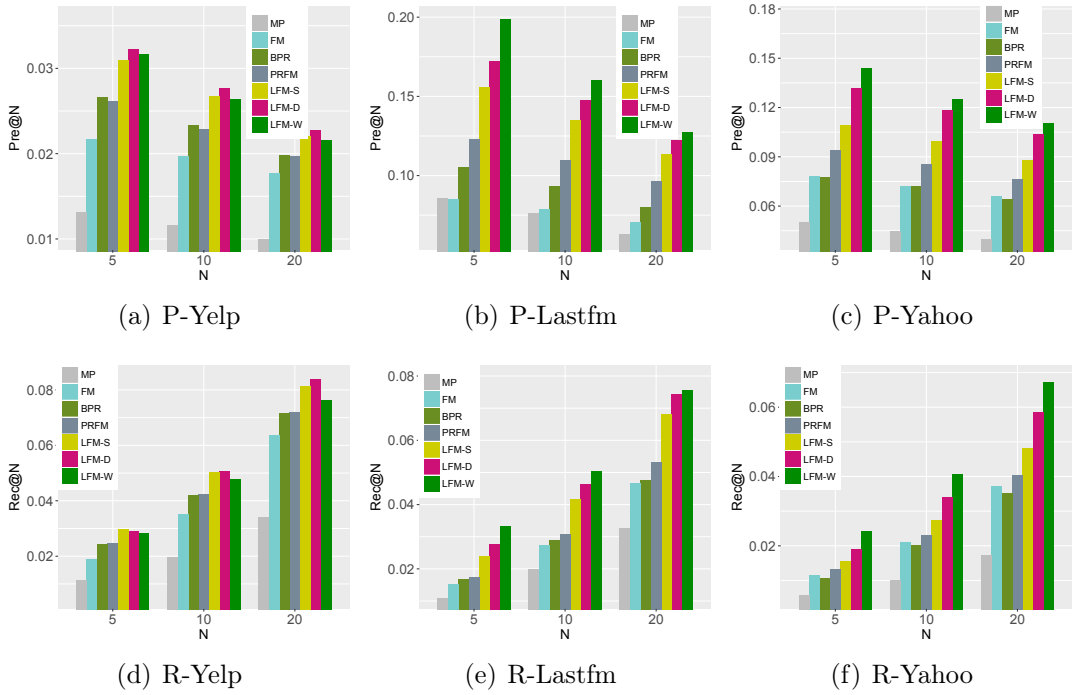
All experiments are conducted with the standard 5-fold cross validation. The average results over 5 folds are reported as the final performance.

3.6.2.1 Accuracy Summary

Table 3.2 and Figure 3.3(a-f) show the performance of all the methods on the three datasets. Several insightful observations can be made: First, in most cases personalized models (BPR, PRFM, LambdaFM) noticeably outperform MP, which is a non-personalized method. This implies, in practice, when there is personalized information present, personalized recommenders are supposed to outperform non-personalized ones. Particularly, our LambdaFM clearly outperforms all the counterparts in terms of four ranking metrics. Second, we observe that different models yield basically consistent recommendation accuracy on different metrics, except for AUC, which we give more details later. Third, as N increases, values of Pre@N get lower and values of Rec@N become higher. The trends reveal the typical behavior of recommender systems: the more items are recommended, the better the recall but the worse the precision achieved.

Regarding the effectiveness of factorization models, we find that BPR performs better than FM on the Yelp dataset[1], which empirically implies pairwise methods

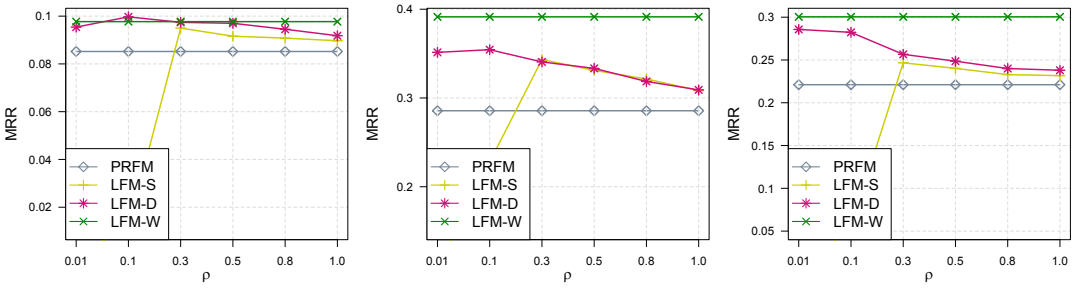
[1] Note that it is not comparable on the Lastfm and Yahoo datasets since FM combines other contexts.



(a) P-Yelp (b) P-Lastfm (c) P-Yahoo

(d) R-Yelp (e) R-Lastfm (f) R-Yahoo

Figure 3.3: Performance comparison w.r.t. top-N values, i.e., Pre@N (P) and Rec@N (R).



(a) Yelp (b) Lastfm (c) Yahoo

Figure 3.4: Parameter tuning w.r.t. MRR.

outperform pointwise ones for ranking tasks with the same negative sampling scheme. The reason is that FM is identical to matrix factorization with only context-item information. In this case, the main difference of FM and BPR is that they apply different loss functions, i.e., quadratic and logistic loss, respectively. This is in accordance with the interesting finding that BPR and PRFM produce nearly the same results w.r.t. all metrics on Yelp. Furthermore, among all the

baseline models, the performance of PRFM is very promising. The reason is that PRFM, (1) as a ranking-based factorization method, is more appropriate for handling top-N item recommendation tasks (vs. FM); (2) by applying FM as scoring function, PRFM estimates more accurate ordering relations than BPR due to auxiliary contextual variables.

LambdaFM vs. PRFM: We observe that in Figure 3.3 and Table 3.2 our LambdaFM (including LFM-S, LFM-D and LFM-W) consistently outperforms the state-of-the-art method PRFM in terms of four ranking metrics. In particular, the improvements on Lastfm and Yahoo datasets w.r.t. NDCG and MRR, are more than 6% and 35%, respectively. Similar improvements w.r.t. Pre@N and Rec@N metrics can be observed from Figure 3.3. For clarity, we only use MRR for the following discussion since the performance trend on all other ranking metrics is highly consistent. Besides, we observe that LFM underperforms PRFM in terms of AUC. The results validate our previous analysis, namely, LambdaFM is trained to optimize ranking measures while PRFM aims to maximize the AUC metric. This indicates that pairwise ranking models with the uniform negative sampling strategy, such as BPR and PRFM, may not be good approximators of the ranking biased metrics and such a mismatch empirically leads to non-optimal ranking. From the sampling perspective, the uniform sampler used in PRFM may spends too many efforts on training uninformative examples and learns less useful information than the lambda-based negative sampling strategies.

3.6.2.2 Effect of Lambda Surrogates/Samplers

In this subsection, we study the effect of lambda surrogates (i.e., the sampling distribution) on the recommendation performance. For LFM-S, we directly tune the value of ρ; for LFM-D, we fix the number of sampling units m to a constant, e.g., $m = 10$, and then tune the value of $\rho \in \{0.01, 0.1, 0.3, 0.5, 0.8, 1.0\}$; and for LFM-W the only parameter ε is fixed at 1 for all three datasets in this work. Figure 3.4 depicts the performance changes by tuning ρ. First, we see LFM-W performs noticeably better than PRFM, and by choosing a suitable ρ (e.g., 0.3 for LFM-S and 0.1 for LFM-D), both LFM-D and LFM-S also perform largely better than PRFM. This indicates the three suggested lambda-based samplers work effectively for handling the mismatch drawback between PRFM and ranking

measures. Second, LFM-S produces best accuracy when setting ρ to 0.3 on all datasets but the performance experiences a significant decrease when setting ρ to 0.1. The reason is that the SGD learner will make more gradient steps on popular items due to the oversampling when ρ is set too small according to Eq. (3.11). In this case, most less popular items will not be picked for training, and thus the model is under-trained. This is why in Rendle and Freudenthaler (2014), the authors claimed that popularity-based sampler underperforms the uniform sampler. By contrast, the performance of LFM-D has not dropped on the Yahoo dataset even when ρ is set to 0.01 (which means picking the top from m randomly selected items, see Algorithm 2). This implies that the performance may be further improved by setting a larger m. Third, the results indicate that LFM-D and LFM-W outperform LFM-S on Lastfm and Yahoo datasets[1]. This is because the static sampling method ignores the fact that the estimated ranking of a non-positive item j changes during learning, i.e., j might be ranked high in the first step but it is ranked low after several iterations[2]. Besides, LFM-S computes the item ranking based on the overall popularity, which does not reflect the context information. Thus, it gives more benefit by exploiting current context to perform sampling. The effect of ρ indicates that sampling distribution of negative items significantly impacts the recommendation accuracy. Clearly, according to the above discussions, it can be seen that our thesis **statement (1)** and **statement (2)** are well supported.

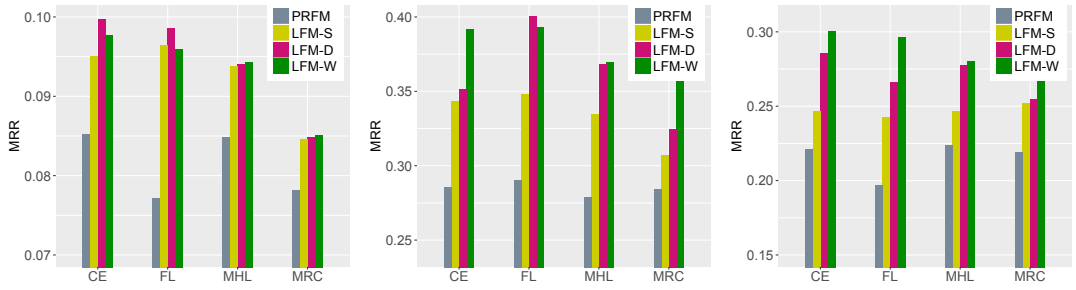
3.6.2.3 Effect of Adding Features

Finding effective context or item content is not the main focus of the thesis but it is interesting to see to what extent LambdaFM improves the performance by adding some additional features. In this section, we study the capability of LambdaFM in modeling content/context features. Therefore, we conduct a contrast experimentation and show the results on Figure 3.6(a-b)[3], where (c, i) denotes a context-item (i.e., music) pair and (c, i, a) denotes a context-item-artist

[1] The superiority is not obvious on the Yelp dataset. This might be due to the reason that the Yelp dataset has no additional context and the item tail is relatively shorter than that of other datasets.

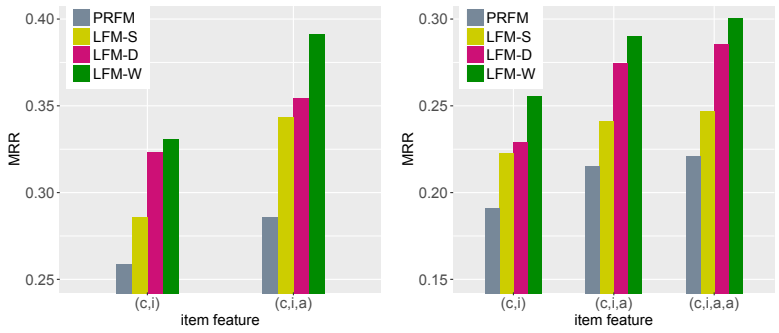
[2] In spite of this, non-positive items drawn by popularity sampler are still more informative than those drawn by uniform sampler.

[3] ρ is assigned a fixed value for the study of context effect, namely 0.1 and 0.3 for LFM-D and LFM-S, respectively.



(a) Yelp (b) Lastfm (c) Yahoo

Figure 3.5: The variants of PRFM and LambdaFM based on various pairwise loss functions.



(a) Lastfm (b) Yahoo

Figure 3.6: Performance comparison w.r.t. MRR with different item features.

triple on both datasets; similarly, (c, i, a, a) represents a context-item-artist-album quad on Yahoo dataset. First, we observe LambdaFM performs much better with (c, i, a) tuples than that with (c, i) tuples on both datasets. This result is consistent with the intuition that a context may like a song if he/she likes another song by the same artist. Second, as expected, LambdaFM with (c, i, a, a) tuples performs further better than that with (c, i, a) tuples from Figure 3.6(b). We draw the conclusion that, by inheriting the advantage of FM, the more effective feature incorporated, the better LambdaFM performs.

3.6.2.4 Lambda with Alternative Loss Functions

Following the implementations of well-known pairwise LtR approaches introduced in Section 3.5, we have built a family of PRFM and LambdaFM variants. Fig-

ure 3.5 shows the performance of all these variants based on corresponding loss functions. Based on the results, we observe that by implementing the lambda surrogates, all variants of LambdaFM outperform PRFM, indicating the generic properties of these surrogates. Among the three lambda surrogates, the performance trends are in accordance with previous analysis, i.e., LFM-W and LFM-D perform better than LFM-S. Additionally, most recent literature using pairwise LtR for recommendation is based on BPR optimization criterion, which is equivalent to the CE loss in the implicit feedback scenarios. To the best of our knowledge, the performance of FL and MHL loss functions have not been investigated in the context of recommendation, and also PRFM and LambdaFM based on FL and MHL loss functions achieve competitive performance with the ones using the CE loss. We expect our suggested PRFM (with new loss functions) and LambdaFM to be valuable for existing recommender systems that are based on pairwise LtR approaches.

3.7 Chapter Summary

In this chapter, we have presented a novel ranking predictor Lambda Factorization Machines (LambdaFM). LambdaFM is a negative sampling based algorithm, although it is analyzed from the top-N ranking perspective. In contrast with PRFM, LambdaFM has more advanced samplers, which can effectively find informative negative examples. LambdaFM is unique in two aspects: (1) it is capable of optimizing various top-N item ranking metrics in implicit feedback settings; (2) it is very flexible to incorporate various context information for context-aware recommendations. Different from the original lambda strategy, which is tailored for the CE loss function, we have proved that the proposed sampling surrogates are more general and applicable to a set of well-known ranking loss functions. Furthermore, we have built a family of PRFM and LambdaFM algorithms, shedding light on how they perform in real tasks. In our evaluation, we have shown that LambdaFM largely outperforms the state-of-the-art counterparts in terms of four standard ranking measures. The methodologies and conclusions in this chapter support the thesis **statement (1)** and **statement (2)**.

The intuition and sampling design of LambdaFM are also applicable to several

other research fields with positive-only data, e.g., the word embedding and visual semantic embedding tasks, where user-item relation in the item recommendation task can be regarded as word-word relation in the word embedding (and image-class relation in the visual semantic embedding) task. Understanding this, it is possible to adapt the model from one domain to another with some slight changes. For example, in Guo et al. (2018a), we successfully adapt the ranking and negative sampling method in LambdaFM for the word embedding task; and in Guo et al. (2018b), we observe that the adaptive sampler proposed in Rendle and Freudenthaler (2014) for item recommendation may also be adapted to improve the repeated sampling process in Weston et al. (2011) for image recognition, although Weston et al. (2011) is not based on the BPR loss which is the main claim in Rendle and Freudenthaler (2014). The main reason that a specific model can be used in very different scenarios is probably because data in the three research fields has some similar distribution. But note that the algorithm may perform slightly different since characteristics of these datasets are not exactly the same. Empirically, the data in the visual semantic embedding task is much sparser than that in the word embedding task. Moreover, different learning algorithms are also impacted differently by these sampling methods. Here, we intend to clarify the similarity and difference between these works. Our insightful observation potentially suggests that many specific models developed in one of these fields are promising to benefit others by minor (or no) changes. We believe this will open a new direction of research to bridge these fields (Yuan et al., 2018b).

Chapter 4

Boosting Factorization Machines

In this chapter, we design an ensemble method that applies LambdaFM (Yuan et al., 2016b) and PRFM (Qiang et al., 2013) as component recommenders, called Boosting Factorization Machines (BoostFM). From this perspective, BoostFM is also a negative sampling based model for implicit feedback scenario. BoostFM combines the strengths of boosting and factorization machines during the process of item ranking. Specifically, BoostFM is an adaptive boosting framework that linearly combines multiple homogeneous component recommenders, which are repeatedly constructed on the basis of individual FM model by a re-weighting scheme. To demonstrate its effectiveness, we perform experiments on three publicly available datasets and compare BoostFM (with uniform and static sampling) to state-of-the-art baseline models.

This chapter is mainly based on our previous work "BoostFM: Boosted Factorization Machines for Top-N Feature-based Recommendation" (Yuan et al., 2017) published in The 22nd Annual Meeting of The Intelligent User Interfaces Community (IUI) 2017 with DOI: http://dx.doi.org/10.1145/3025171.3025211.

4.1 Introduction

Ensemble learning has become a prevalent method to boost machine learning results by combining several models. In this chapter, we make contributions on ensemble-based recommendation models. Specifically, we apply boosting techniques to improve Factorization Machines (FM) in implicit feedback scenarios. Boosting techniques were first employed to improve the performance of classification by integrating a set of weak classifiers (i.e., the classification accuracy

rate should larger than 0.5) into a stronger one with better performance (Jiang et al., 2013). Note that since the employed component recommenders all perform significantly better than random guessing, we do not specify this requirement. Previous research has proven that boosting techniques usually come with better convergence properties and stability (Bertoni et al., 1997; Chowdhury et al., 2015). So far, the most common implementation of boosting is AdaBoost (Freund and Schapire, 1997), although some newer boosting variants are reported (Freund, 2001; Freund et al., 2003; Xu and Li, 2007). We find that boosting techniques have been recently introduced to solve recommendation problems with better results than single collaborative filtering (CF) algorithms (Jiang et al., 2013; Wang et al., 2014; Liu et al.; Chowdhury et al., 2015). However, all existing solutions are based on the basic matrix factorization model, which fails to incorporate more general context information. Moreover, in our work the learning process of each component recommender is optimized for top-N recommendation with implicit feedback, which is different from most previous work either optimized for rating prediction (Jiang et al., 2013) or even optimized for ranking but on explicit rating datasets (Chowdhury et al., 2015; Wang et al., 2014).

In this chapter, we propose BoostFM, a boosting approach for top-N context-aware CF, by combining the most well-known boosting framework AdaBoost with FM. Specifically, we first choose Factorization Machines to build the component recommender and multiple homogeneous component recommenders are linearly combined to create a strong recommender. The coefficient of each component recommender is calculated from the weight function based on a certain performance metric. At each boosting round, we devise a ranking objective function to optimize the component recommender following PRFM and LambdaFM. That is each component recomender in BoostFM is also based on negative sampling method. In addition, in the process of learning, we develop a re-weighting strategy and assign a dynamic weight to force the optimization concentrate more on observed context-item interactions with bad evaluation performance.

4.2 Related Work about Boosting

Boosting is a general technique for improving the accuracy of a given learning algorithm (Freund and Schapire, 1997; Xu and Li, 2007). The basic idea is to repeatedly construct a number of 'weak learners' by using the homogeneous weak algorithm on re-weighting training data. Then, a strong learner with boosted total performance is created by composing weak learners linearly. Boosting was originally developed to enhance the performance of binary classification, where AdaBoost (Adaptive Boosting) is the most well-known boosting algorithm. Following this, various extensions have been made to deal with problems of multi-class classification (Friedman et al., 2000), regression (Bertoni et al., 1997), and ranking (Xu and Li, 2007).

Recently, researchers have proposed conducting the boosting technique in Recommender Systems. For example, two boosting frameworks based on AdaBoost have been proposed for the rating prediction task by applying both memory- and model-based CF algorithms (Jiang et al., 2013). AdaMF (Wang et al., 2014) borrows the idea from adaRank by combining matrix factorization (MF) recommender with boosting methods. The coefficient function for each MF recommender is calculated based on the Normalized Discount Cumulative Gain (NDCG) performance of the stronger recommender. However, the component recommenders are constructed using the CF algorithm for rating prediction, which is suboptimal for item recommendation task in the same setting (such as the sampling method). Similar work has been done in (Chowdhury et al., 2015), where the component recommender is constructed using probability matrix factorization (PMF) on explicit rating datasets.

Our work is related to above work, but differ in several significant differences. First, in BoostFM, the component recommender is constructed by feature-based FM model instead of a simple approach (i.e., so-called weak learner). Note that individual FM model can easily achieve good prediction performance. In this work, we can regard FM as a relatively strong[1] recommender. Second, the component recommender is constructed by optimizing a weighted ranking metric (i.e., AUC)

[1] Previous literature has shown that AdaBoost demonstrates better generalizing performance with correlated strong learners (Li et al., 2008).

with implicit feedback, not approximating users' explicit ratings (e.g., Jiang et al. (2013); Wang et al. (2014)). In addition, in the boosting procedure, each observed context-item pair, is treated as a training instance for the weighting calculation, which differs from previous work treating a given user (Wang et al., 2014) (or query (Xu and Li, 2007)) as a training instance. BoostFM is the first study for feature-based collaborative ranking by utilizing the boosting technique. Note it is worth mentioning that one recent work (Cheng et al., 2014) have exploited the gradient boosting algorithm for content/context-based rating prediction with FM. However, this work targeted at the feature selection procedure with gradient boosting technique, which is completely different from our work concentrating on improving top-N recommendation.

4.3 Preliminaries

For better understanding the framework of BoostFM, we briefly restate the problem of content/context-based recommendation based on implicit feedback in this section.

Let $S \subset C \times I$ be a set of observed interactions (i.e., so-called implicit feedback), where C is a set of context and I a set of items. For example, C could be a set of users, and I a set of music, and S denotes which music tracks a user has played, i.e., (a set of) user-music pairs. As previously mentioned, C can handle more complex examples with additional variables, e.g., mood, social friends, time and locations, also I might express item with additional side information, e.g., artist and albums. The task of content/context-based recommendation is to find a ranking $\hat{r}$ of items I for each context c, which can be formulated by a bijective function (Rendle and Freudenthaler, 2014), i.e., $\hat{r}$: $I \times C \rightarrow \{1, ..., |I|\}$. $\hat{r}(j|c)$ is the ranking of item j under given context c, which is usually modeled by a scoring function $\hat{y}(j|c)$ (i.e., the FM model in this chapter).

For the item recommendation task, the accuracy of a recommender is usually evaluated using various ranking metrics, such as AUC[1] (Rendle et al., 2009b), the Normalized Discount Cumulative Gain (NDCG) (Pan and Chen, 2013), Precision@N and Recall@N (Li et al., 2015b). For example, the definitions of AUC

[1]Note that maximizing a smoothed AUC is still a popular way for optimizing a ranking algorithm (e.g., (Shi et al., 2014; Rendle and Freudenthaler, 2014; Zhao et al., 2014)), although it is position-independent.

and NDCG per context is given below.

$$\text{AUC(c)} = \frac{1}{|I_c^+|} \sum_{i \in I_c^+} \frac{1}{|I \backslash I_c^+|} \sum_{j \in I \backslash I_c^+} \mathbb{I}(\hat{r}(i|c) < \hat{r}(j|c)) \tag{4.1}$$

$$\text{NDCG(c)} = Z_c \sum_{\hat{r}(i|c)=1}^{|I|} \frac{2^{\text{rel}_{\hat{r}(i|c)}} - 1}{\log_2 \left(\hat{r}(i|c) + 1\right)} \tag{4.2}$$

where I_c^+ represents the set of items that have been selected under context c, and $\mathbb{I}(\cdot)$=1 if the condition is true, and 0 otherwise; $rel_{\hat{r}}$ represents the relevance score of a candidate item at the position $\hat{r}$, here we use a binary value 0-1 for quantity. Z_c is calculated from the normalization constant so that the ideal ranker will get NDCG of 1.

4.4 Boosted Factorization Machines

We propose a novel algorithm to solve the context-aware recommendation problem by optimizing the ranking measures. The algorithm is referred to as Boosted Factorization Machines (BoostFM for short), the derivation of which is inspired by AdaBoost, AdaRank (Xu and Li, 2007) and and Liu et al.; Jiang et al. (2013); Wang et al. (2014); Chowdhury et al. (2015). While AdaRank is not suitable for sparse data prediction and Liu et al.; Jiang et al. (2013); Wang et al. (2014); Chowdhury et al. (2015) is only suited for the basic collaborative filtering task with two input features. BoostFM is generic boosting algorithm tailed for sparse features and optimized by the ranking loss.

4.4.1 BoostFM

We aim at devising a set of 'weak learner'[1] sequentially to model the pairwise interactions between various feature variables. However, the BoostFM algorithm will be able to concentrate hard on optimizing the objective function defined based on ranking metrics. We observe from Eq. (4.1) and Eq. (4.2) that the accuracy of a recommender model is determined by the rank positions (i.e. $\hat{r}(i|c)$) of positive items $i \in I_c^+$ of each context c. Thus, we devise a general performance measure function $E\left[\hat{r}(c, i, g)\right]$ to denote the recommendation accuracy associated with each

[1] As a 'weak learner', FM fairly meets the basic conditions, reflected in both linear complexity and higher prediction accuracy than random guessing.

observed context-item pair. The argument of general function $\hat{r}(c, i, g)$ is the rank position of item i for each context c, calculated by the trained function g. Thus we can rewrite the ranking metric of AUC and NDCG as below

$$\text{AUC} = \frac{1}{|C|}\sum_{c\in C}\frac{1}{|I_c^+|}\sum_{i\in I_c^+} E\left[\hat{r}(c,i,g)\right] = \frac{1}{|C|}\sum_{(c,i)\in S}\frac{1}{|I_c^+|}E\left[\hat{r}(c,i,g)\right] \tag{4.3}$$

where

$$E\left[\hat{r}(c,i,g)\right] = \frac{1}{|I\backslash I_c^+|}\sum_{j\in I\backslash I_c^+}\mathbb{I}(\hat{r}(i|c) < \hat{r}(j|c)) \tag{4.4}$$

$$\text{NDCG} = \frac{1}{|C|}\sum_{c\in C} Z_c \sum_{\hat{r}(i|c)=1}^{|I|} E\left[\hat{r}(c,i,g)\right] = \frac{1}{|C|}\sum_{(c,i)\in S} Z_c E\left[\hat{r}(c,i,g)\right] \tag{4.5}$$

where

$$E\left[\hat{r}(c,i,g)\right] = \frac{1}{\log_2\left(\hat{r}(i|c)+1\right)} \tag{4.6}$$

To maximize Eq. (4.3) and Eq. (4.5), we propose to minimize the following objective function (Note that $\frac{1}{|C|}$, $\frac{1}{|I_c^+|}$, Z_c are normalizing constants).

$$\underset{g\in\Omega}{\text{argmin}} \sum_{(c,i)\in S}\left\{1 - E\left[\hat{r}(c,i,g)\right]\right\} \tag{4.7}$$

where Ω is the set of ranking scoring functions. It is non-trivial to directly optimize $E\left[\hat{r}(c,i,g)\right]$, which is a non-continuous function. Instead, we propose to minimize an upper bound of Eq. (4.7) by leveraging the property $e^{-x} \geq 1 - x$ ($x \in \mathbb{R}$) such that it can be fitted in AdaBoost Framework easily.

$$\underset{g\in\Omega}{\text{argmin}} \sum_{(c,i)\in S}\exp\{-E\left[\hat{r}(c,i,g)\right]\} \tag{4.8}$$

Following the idea of AdaBoost, BoostFM is expected to generate a strong recommender by linearly combining multiple homogeneous component recommenders[1]. Thus the ranking function (so-called strong recommender) g can be expressed as

$$g^{(t)} = \sum_{t=1}^{T}\beta_t\hat{y}^{(t)} \tag{4.9}$$

where $\hat{y}^{(t)}$ is the t-th component recommender constructed by the FM model and $\beta_t \in \mathbb{R}^+$ is the coefficient which is usually determined by the overall recommendation performance (e.g., AUC or NDCG) of $\hat{y}^{(t)}$ at t-th boosting round. BoostFM runs for T rounds and creates a new component recommender $y^{(t)}$ at each round.

[1]The term weak learner, weak recommender and component recommender are used interchangeably in this chapter.

Algorithm 4 BoostFM

Input: The observed context-item interactions S, parameters E and T.
Output: The strong recommender $g^{(T)}$
Initialize $\mathbf{Q}_{ci}^{(t)} = 1/|S|$, $g^{(0)} = 0$, $\forall (c,i) \in S$
for t =1,..., T **do**
    Create $\hat{y}^{(t)}$ with $\mathbf{Q}^{(t)}$ on S,$\forall (c,i) \in S$, i.e. Algorithm 5;
    Calculate the ranking accuracy $E\left[\hat{r}(c,i,g)\right]$, $\forall (c,i) \in S$;
    Calculate the coefficient β_t,
    $\beta_t = \ln(\frac{\sum_{(c,i)\in S} \mathbf{Q}_{ci}^{(t)}\{1+E\left[\hat{r}(c,i,y^{(t)})\right]\}}{\sum_{(c,i)\in S} \mathbf{Q}_{ci}^{(t)}\{1-E\left[\hat{r}(c,i,y^{(t)})\right]\}})^{\frac{1}{2}}$;
    Create the strong recommender $g^{(t)}$,
    $g^{(t)} = \sum_{h=1}^{t} \beta_h \hat{y}^{(h)}$;
    Update weight distribution $\mathbf{Q}^{(t+1)}$,
    $\mathbf{Q}_{ci}^{(t+1)} = \frac{\exp\{-E\left[\hat{r}(c,i,g^{(t)})\right]\}}{\sum_{(c,i)\in S} \exp\{-E\left[\hat{r}(c,i,g^{(t)})\right]\}}$
end for

Then the newly trained recommender is integrated to the final ensemble recommender $g^{(t)}$. The minimization in Eq. (4.8) is converted to

$$\arg \min_{\beta_t, y^{(t)} \in \Phi} \sum_{(c,i)\in S} \exp\{-E\left[\hat{r}(c,i,g^{(t-1)} + \beta_t y^{(t)})\right]\} \tag{4.10}$$

where Φ is the set of possible component recommenders., and $g^{(t-1)} = \sum_{h=1}^{t-1} \beta_h y^{(h)}$. To solve Eq. (4.10), we propose to maintain a distribution of weights over each observed (c,i) pair in the training data, denoted by matrix $\mathbf{Q} \in \mathbb{R}^{|C|\times|I|}$. The weight value on the (c,i) training instance at round t is denoted by $\mathbf{Q}_{ci}^{(t)}$. More specifically, the weight distribution reflects the emphasis on the component recommender. At each boosting round, weight values $\mathbf{Q}_{ci}^{(t)}$ on (c,i) pairs with low rank performance by the ensemble strong recommender (i.e., Eq. (4.9)) is increased so that the component recommender at next boosting round would be forced to give more penalties to those 'hard' training instances. For the implementation of BoostFM, we propose to employ the 'forward stage-wise approach' (Li et al., 2015b), where $g^{(t)}$ is treated as the additive model, $y^{(t)}$ is the basis function, and β_t is the expansion coefficient of a basis function. BoostFM starts with $g^{(0)} = 0$, and then adds new basis functions greedily, without changing the parameters (i.e., Θ) and coefficients of those that have already been added. At each round t, a new expansion coefficient β_t and basis function $y^{(t)}$ can be found to minimize the exponential objective function. More details about the BoostFM have been

shown in Algorithm 4. Note that it is computationally expensive to calculate $E\left[\hat{r}(c,i,y^{(t)})\right]$ directly due to the large size of implicit feedback, we solve it by first performing a uniform sampling to obtain a few non-observed items (say 50), and then calculating the rank of i among them as an unbiased estimator of $\hat{r}(i|c)$. Following AdaRank (Xu and Li, 2007), it can be proved that as follows there exists a lower bound in terms of the performance measures can be continuously grown.

Theorem 1. *The following bound holds in terms of ranking accuracy (e.g., AUC) of BoostFM algorithm on the training data:*

$$\frac{1}{|C|}\sum_{(c,i)\in S}\frac{1}{|I_c^+|}E\left[\hat{r}(c,i,g)\right] \geq \frac{\sum_{(c,i)\in S}\frac{1}{|I_c^+|}}{|C|}\left[1-\prod_{t=1}^{T}e^{-\delta_{min}^t\sqrt{1-\pi(t)^2}}\right]$$

where $\pi(t)=\sum_{(c,i)\in S}\boldsymbol{Q}_{ci}^{(t)}E\left[\hat{r}(c,i,y^{(t)})\right]$, $\delta_{min}^t=min_{(c,i)\in S}\delta_{ci}^t$, *and* $\delta_{ci}^t=E\left[\hat{r}(c,i,g^{(t-1)}+\beta_t y^{(t)})\right]-E\left[\hat{r}(c,i,g^{(t-1)})\right]-\beta_t E\left[\hat{r}(c,i,y^{(t)})\right]$, *for all* $(c,i)\in S$ *and* $t=1,2,...,T$.

4.4.2 Component Recommender

Since this work targets at the top-N recommendation task, we thus propose the ranking optimization methods to create the component recommenders. Naturally, it is feasible to exploit the Learning-to-Rank (LtR) techniques to optimize Factorization Machines (FM). As has been introduced in Chapter 3, In the following, we present BoostFM with both PRFM and LambdaFM as component recommenders.

4.4.2.1 Weighted Pairwise Factorization Machines

Following PRFM in Chapter 3, We employ the weighted cross entropy (CE) as the pairwise objective function to learn the preference relations between each two (c,i) pairs. The component recommender is referred to as Weighted Pairwise Factorization Machines (WPFM for short).

By combining the weight distribution of $\mathbf{Q}$, the objective function and gradient for each $(i,j)_c$ pair is given by

$$L\big((i,j)_c\big)=\mathbf{Q}_{ci}^{(t)}\log(1+\exp(-\sigma(\hat{y}(\mathbf{x}^i)-\hat{y}(\mathbf{x}^j)))) \tag{4.11}$$

Algorithm 5 Component Recommender Learning

Input: The set of all pairwise relations D_s, the weight distribution $\mathbf{Q}^{(t)}$, regularization parameters γ_θ, learning rate η
Output: Parameters $\Theta = (\mathbf{w}, \mathbf{V})$
Initialize Θ: $\mathbf{w} \leftarrow (0, ..., 0)$; $\mathbf{V} \sim \mathcal{N}(0, 0.1)$;
repeat
    Draw (c, i) from S uniformly ;
    Draw j from $I \backslash I_c^+$ uniformly ;
    for $f \in \{1, ..., k\}$ **do**
        for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
            Update $v_{d,f}$;
        end for
    end for
    for $d \in \{1, ..., n\} \wedge x_d \neq 0$ **do**
        Update w_d;
    end for
until meet the condition (e.g., after several iterations)

$$\frac{\partial L((i,j)_c)}{\partial \theta} = \lambda_{i,j}\left(\frac{\partial \hat{y}(\mathbf{x}^i)}{\partial \theta} - \frac{\partial \hat{y}(\mathbf{x}^j)}{\partial \theta}\right) \tag{4.12}$$

where $\hat{y}(\mathbf{x})$ is the FM model, and $\lambda_{i,j}$ is the learning weight for $(i,j)_c$ pair, given as

$$\lambda_{i,j} = \frac{\partial L((i,j)_c)}{\partial(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))} = -\frac{\sigma \mathbf{Q}_{ci}^{(t)}}{1 + \exp(\sigma\big(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j)\big))} \tag{4.13}$$

Following the similar derivation in Chapter 3, we have

$$w_d \leftarrow w_d - \eta(\lambda_{i,j}(x_d^i - x_d^j) + \gamma_{w_d} w_d) \tag{4.14}$$

$$v_{d,f} \leftarrow v_{d,f} - \eta(\lambda_{i,j}(\sum_{l=1}^{n} v_{l,f}(x_d^i x_l^i - x_d^j x_l^j) - v_{d,f}({x_d^i}^2 - {x_d^j}^2)) + \gamma_{v_{d,f}} v_{d,f})$$

where γ_θ (i.e., γ_{w_d}, $\gamma_{v_{d,f}}$) is a hyper-parameter for the L2 regularization, and η is the learning rate. To deal with the large number of unobserved feedback (i.e., $I \backslash I_c^+$), the common practice is to exploit Stochastic Gradient Descent (SGD) with uniform sampling. Finally, Algorithm 5 shows how the component recommender is optimized with weighted pairwise learning.

4.4.2.2 Weighted LambdaFM Factorization Machines

As mentioned in LambdaFM, the static sampler shows better performance than the uniform sampler. Hence, we employ LambdaFM as a component recom-

Table 4.1: Basic statistics of datasets. Each tuple represents an observed context-item interaction. Note that tags on the MLHt dataset are regarded as recommended items (i.e., i in $\mathbf{x}_i$), while a user-item (i.e., user-movie) pair is regarded as context (i.e., c in $\mathbf{x}_c$).

Datasets	Users	Items	Tags	Artists	Albums	Tuples
MLHt	2113	5908	9079	-	-	47958
Lastfm	983	60000	-	25147	-	246853
Yahoo	2450	124346	-	9040	19851	911466

mender, the way of which is referred to as Weighted Lambda Factorization Machines (WLFM).

In Chapter 3, we have proposed three lambda-based negative sampler. In BoostFM, we only investigate the static sampler since it is does not have additional computational complexity. However, we find that a following work by Li et al. (2018) inspired LambdaFM and BoostFM has verified all three samplers and show consistent performance. Here, we use the same static negative sampler in LambdaFM, i.e., sampling more popular items approximately proportional to the empirical popularity distribution. p_j is given below

$$p_j \propto \exp(-\frac{r(j)}{|I| \times \rho}), \rho \in (0, 1] \tag{4.15}$$

where $r(j)$ represents the rank of item j among all items I according to the overall popularity, ρ is a parameter to control the sampling distribution of negative items. Therefore, Line 6 in Algorithm 5 can be replaced by the above sampler.

4.5 Experiments

In this section, we conduct experiments on the three real-world datasets to verify the effectiveness of BoostFM in various settings.

4.5.1 Experimental Setup

4.5.1.1 Datasets

We use three publicly accessible recommendation datasets for our experiments, namely, MovieLens Hetrec (MLHt)[1] (user-movie-tag triples, where the context is a user-movie pair, the item is the tag), Lastfm[2] (user-music-artist triples, where

[1] http://grouplens.org/datasets/hetrec-2011/
[2] dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html

the context is the user, the item is a music track with an artist) and Yahoo music[1] (user-music-artist-album tuples, where the context is the user, the item is a music track with an artist and album). In the MLHt dataset, the task is to recommend top-N relevant tags for each user-movie pair, while on the Lastfm and Yahoo datasets, it is to recommend top-N preferred music tracks (with item side information) to each user. To speed up the experiments, we follow the common practice as in (Christakopoulou and Banerjee, 2015) by randomly sampling a subset of users from the user pool of the Yahoo dataset[2], and a subset of items from the item pool of the Lastfm dataset. The MLHt dataset is kept in its original form. The statistics of the datasets after preprocessing are summarized in Table 4.1.

4.5.1.2 Evaluation Metrics

To evaluate the performance of BoostFM, we display our results with two widely used ranking metrics, namely, Precision@N and Recall@N (denoted by Pre@N and Rec@N respectively), where N is the number of recommended items (again, tags are considered as items on the MLHt datasets. Please note that the results on other ranking metrics, such as NDCG and MRR, are highly consistent. The definitions of Pre@N and Rec@N have been given in Chapter 3.

4.5.1.3 Baseline Methods

In our experiments, we compare our algorithm with several powerful baseline methods, namely, Most Popular (MP), User-based Collaborative Filtering (UCF), Bayesian Personalized Ranking (BPR), Factorization Machines (FM). Specifically, for tag recommendation on the MLHt dataset, we utilize MP, FM and PITF as baselines. For music recommendation based on additional content information, we utilize MP, UCF, FM, BPR, and PRFM as baselines. For clarity, we refer to BoostFM with WPFM and WLFM as B.WPFM and B.WLFM respectively. We refer to PRFM with the CE loss and Hinge loss as PRFM.CE and PRFM.H respectively. The descriptions of CE and Hinge loss, and baselines, including MP, BPR and FM, have been given in Chapter 3.

[1] http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=2

[2] The number of context-item interactions on the original dataset has close to billion level.

- User-based Collaborative Filtering (UCF) (Gao et al., 2013): It is a typical memory-based CF algorithm applicable for both rating prediction and item ranking tasks. Pearson correlation is used in this work to compute user similarity and the top-20 most similar users are selected as the nearest neighbors.

- Pairwise Interaction Tensor Factorization (PITF) (Rendle and Schmidt-Thieme, 2010): PITF is a state-of-the-art tensor factorization model optimized by the BPR loss. It is the winner in Tag Recommendation of ECML PKDD Discovery Challenge[1].

4.5.1.4 Hyper-parameter Settings

There are several critical hyper-parameters needed to be set for BoostFM.

- The number of component recommender T: For the purpose of comparison, T of BoostFM is set to 10 in all three datasets if not explicitly declared. The contribution of T is discussed later in Section 4.5.2.2.

- Learning rate η and regularization γ_θ: We first employ the 5-fold cross validation to find the best η by running BoostFM with $\eta \in \{0.005, 0.01, 0.02, 0.05, 0.08, 0.1, 0.2, 0.4\}$, and then tune γ_θ the same way by fixing η. Specifically, η is set to 0.08 on the Lastfm and Yahoo datasets, and 0.4 on the MLHt dataset; γ_θ is set to 0.05, 0.02 and 0.005 on the Lastfm, Yahoo and MLHt dataset respectively. In our experiment, we find all FM based models perform well enough by just employing polynomial term (refer to Eq. (2.7)), and thus we omit the configuration of the linear term. Baseline algorithms are tuned in the same way.

- Latent dimension k: Like in Chapter 3, for comparison purposes, the approaches assign a fixed k value (e.g., $k = 30$ in our experiments) for all methods based on factorization models. Results for $k = 10, 50, 100$ show similar behaviors.

- Distribution coefficient ρ: $\rho \in (0, 1]$ tuned according to the data distribution. Details will be given in the later section.

[1] http://www.kde.cs.uni-kassel.de/ws/dc09

4.5.2 Performance Evaluation

All experiments are conducted with the standard 5-fold cross validation. The average results over 5 folds are reported as the final performance.

4.5.2.1 Accuracy Summary

Figure 4.1(a-f) shows the prediction quality of all algorithms on the three datasets. Like in Chapter 3 there are several similar interesting observations that can be made. In this chapter, we mainly focus on investing the performance impact by using the boosting strategy.

BoostFM vs. PRFM and PITF: In Figure 4.1, we observe that our BoostFM (i.e., B.WPFM and B.WLFM) consistently outperforms the state-of-the-art methods PITF and PRFM. For example, on the MLHt dataset, we can calculate that B.WPFM outperforms PITF by 6.1% and 5.4% in terms of Pre@10 and Rec@10 respectively[1]. In particular, the significant improvements by B.WPFM (compared with PRFM.CE and PRFM.H) are more than 18% on Pre@10 and 35% on Rec@10 on both Lastfm and Yahoo datasets. The results shows that the accuracy of top-N recommendation can be largely improved by using boosting technique. Note that B.WPFM will reduce to PITF and PRFM when the component recommenders $T = 1$.

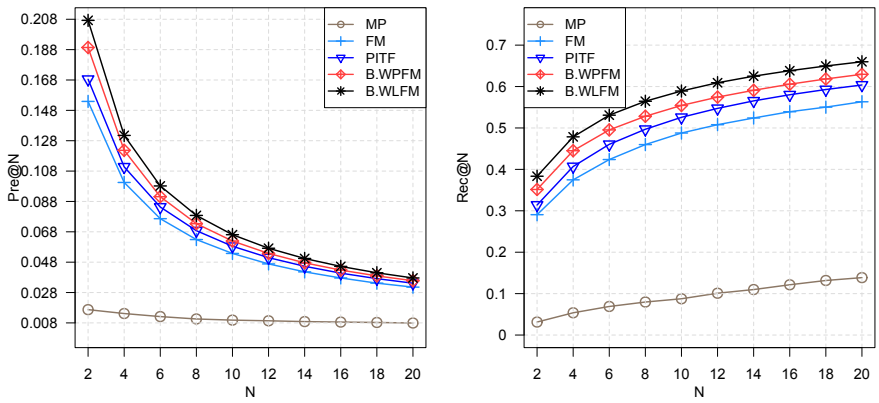
B.WPFM vs. B.WLFM: In contrast to B.WPFM, B.WLFM achieves much better results on all datasets in Figure 4.1. The difference is that the component recommender WLFM is trained by more advanced static sampler while WPFM is trained by a uniform sampler. The impact of different negative samplers have been thoroughly studied in Chapter 3, and the results are consistent with previous studies in LambdaFM.

4.5.2.2 Effect of Number of Component Recommenders

In this subsection, we evaluate the performance sensitivity of BoostFM to the number of component recommenders T. T is adjusted from 1 to 50 and ρ for B.WLFM is set to a fixed value[2] (e.g., 0.3) for a fair comparison. The results

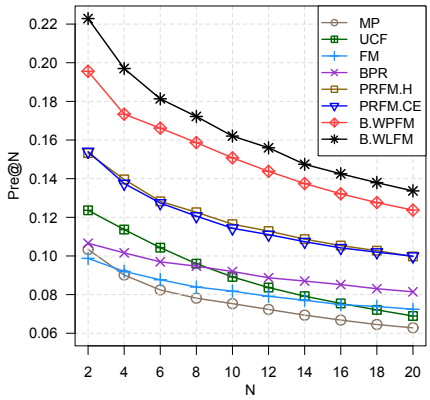
[1] We only use the top-10 (i.e., Pre@10 and Rec@10) value for the following descriptions since the performance trend on other top-N values is consistent.

[2] Again, the performance trend keeps consistent for any value of $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$.

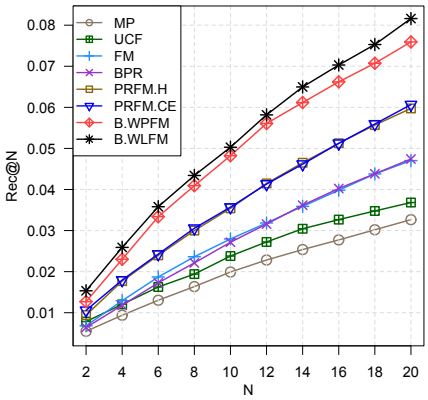


(a) P-MLHt

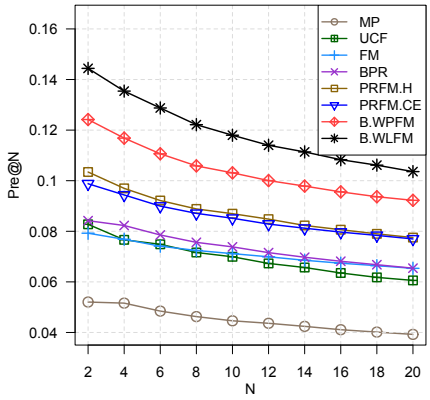
(b) R-MLHt



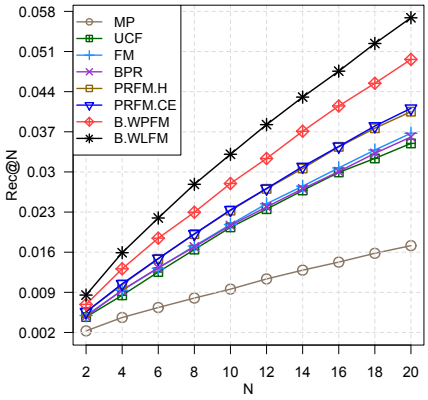
(c) Lastfm



(d) Lastfm

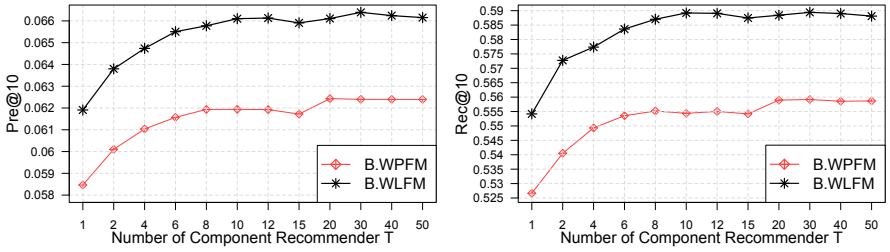


(e) P-Yahoo



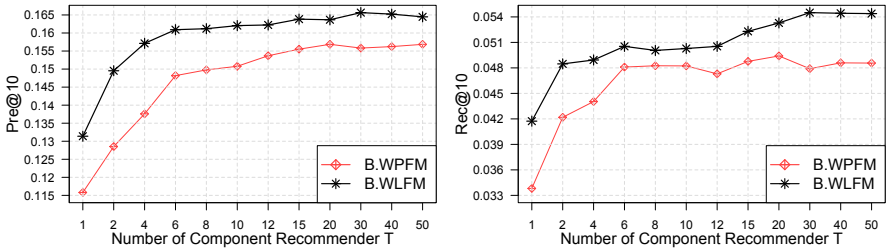
(f) R-Yahoo

Figure 4.1: Performance comparison w.r.t., top-N values, i.e., Pre@N and Rec@N. N ranges from 2 to 20, the number of component recommender T is fixed to 10, and ρ for B.WLFM is fixed to 0.3.



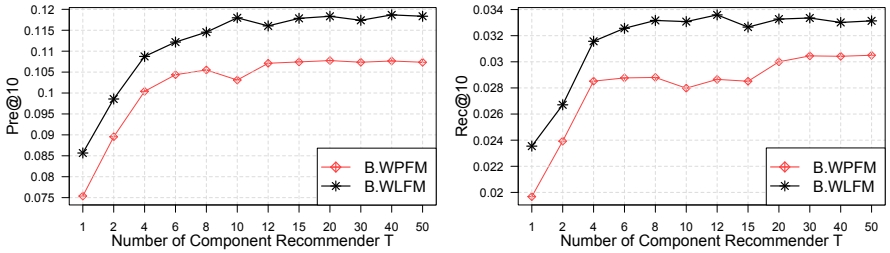
(a) P-MLHt

(b) R-MLHt



(c) Lastfm

(d) Lastfm

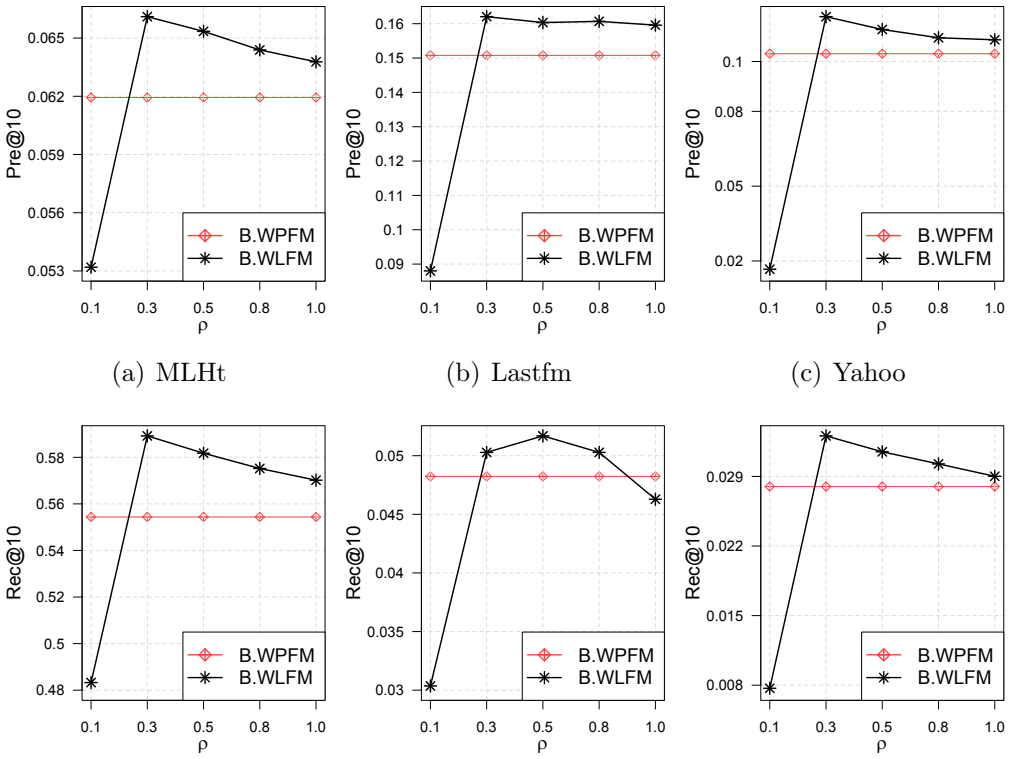


(e) P-Yahoo

(f) R-Yahoo

Figure 4.2: Performance trend of BoostFM (i.e., B.WPFM and B.WLFM) w.r.t. Pre@10 and Rec@10. T ranges from 1 to 50, and ρ is fixed to 0.3.

on all datasets are summarized in Figure 4.2, in terms of Pre@10 and Rec@10. We observe that the top-10 recommendation performance generally improves by increasing the number of component recommenders T, particularly when T is smaller than 10. When T is larger than 10, adding more component recommenders generates marginal performance improvements. In addition, we can observe that B.WLFM performs significantly better than B.WPFM, which is consistent with previous results in Figure 4.1. Again, when T is set to 1, B.WPFM reduces to PRFM.CE. The different performance between them comes from their parameter settings since the best hyper-parameters of B.WPFM are found based on $T = 10$.



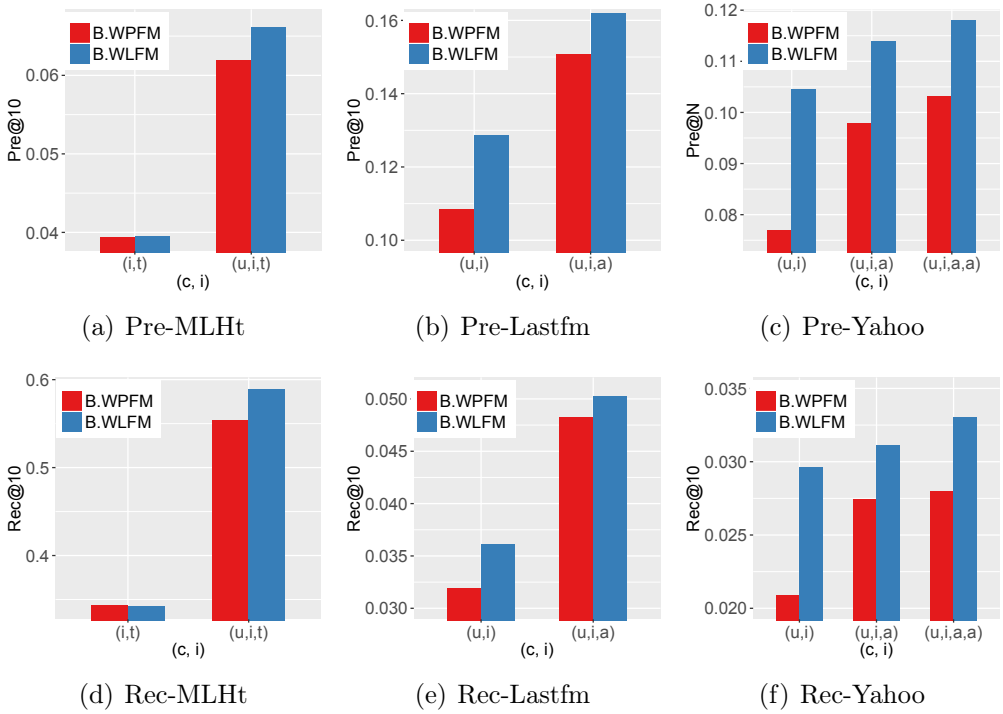
(a) MLHt (b) Lastfm (c) Yahoo

(d) Lastfm (e) Lastfm (f) R-Yahoo

Figure 4.3: Performance trend of BoostFM by tuning ρ w.r.t. Pre@10 & Rec@10. $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$, $T = 10$.

4.5.2.3 Effect of Sampling Strategies (i.e., ρ)

The effect of parameter ρ has been studied in LambdaFM in Chapter 3. Here, we verify it in the setting of BoostFM. Similarly as in Chapter 3, ρ directly controls the sampling distribution of negative items. As expected, it will have a big impact on the recommendation performance. By tuning $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$, we depict the results in Figure 4.3. First, we clearly see that BoostFM with WLFM performs much better than that with WPFM when $\rho \in [0.3, 1.0]$. Particularly, WLFM produces best accuracy when setting ρ to 0.3 in nearly all datasets, but then the performance experiences a significant decrease when setting it to 0.1. The reason is because the component recommender WLFM concentrates more gradient steps on the most popular items due to the oversampling scheme when ρ is set to a small value (i.e., $\rho = 0.1$) based on Eq. (4.15). In this case, most



(a) Pre-MLHt (b) Pre-Lastfm (c) Pre-Yahoo

(d) Rec-MLHt (e) Rec-Lastfm (f) Rec-Yahoo

Figure 4.4: Performance comparison w.r.t. Pre@10 & Rec@10 with context and side information. In (a) (d), (i, t) denotes an item-tag (i.e., movie-tag) pair and (u, i, t) denotes a user-item-tag triple; in (b)(c)(e)(f) (u, i) denotes a user-item (i.e., user-music) pair and (u, i, a) denotes a user-item-artist triple; similarly, (u, i, a, a) denotes a user-item-artist-album quad. T is fixed to 10, and ρ is fixed to 0.3.

less unpopular items will not be chosen for training, and thus the model is under-trained. In practice, we would suggest to tune the parameter ρ progressively from a larger value (e.g., 1.0) to a smaller one and find the one that performs best in the training set. Empirically, the longer tail a recommendation dataset has, the smaller ρ can be adjusted.

4.5.2.4 Effect of Adding Features

Like in Chapter 3, we investigate the performance of BoostFM by adding context and content information. The results are shown in Figure 4.4. First, we observe BoostFM performs much better with (u, i, a) tuples than that with (u, i) tuples on the Lastfm and Yahoo datasets. This result is intuitive as a user may like a music track if she likes another one by the same artist. Second, as expected,

BoostFM with (u, i, a, a) tuples performs further better than that with (u, i, a) tuples in (c) and (f). The results verify the effectiveness of BoostFM in modeling item attribute information. In addition, similar result can be observed on the MHLt dataset: BoostFM achieves largely better results with (u, i, t) tuples than that with (i, t) tuples. We draw the conclusion that the more effective feature information incorporated, the better BoostFM performs. Interestingly, we find an outlier result on the MHLt dataset where the B.WLFM does not outperform B.WPFM with (i, t) pair. By manual inspection, we find that the tags for each user-movie (i.e., c) pair is highly dominated by the user's preference (e.g., tagging habits) rather than the movie itself. In other words, some users assign the same tags for many different movies. The user bias problem can be well handled when the user information is considered during the pairwise comparison (e.g, BoostFM with (u, i, t) tuples), while B.WLFM usually leads to non-improved results without the user context. The reason might be that the overall popular tags for movies may not be popular tags for user-movie pairs.

4.6 Chapter Summary

In this chapter, we have proposed a novel ranking predictor Boosted Factorization Machines (BoostFM) for item recommendation from implicit feedback. Inheriting advantages from both boosting technique and FM, BoostFM (1) is capable of improving top-N recommendation performance by combining a set of component recommenders; (2) is very flexible to integrate auxiliary information, such as context and item content information, for better recommendation accuracy. Extensive results on three real-world datasets show that the suggested BoostFM (both B.WPFM and B.WLFM) clearly outperforms a bunch of state-of-the-art CF counterparts.

Like LambdaFM and PRFM, BoostFM is also a negative sampling based method, although the main work in this chapter is not to propose new negative sampling techniques.

Chapter 5

Geographical Bayesian Personalized Ranking

In the last two chapters, we have introduced two generic recommender models that can be used for both content/context-aware and basic collaborative filtering settings. In this chapter, we focus on introducing a point-of-interest (POI) recommendation approach that explicitly models users' geographical preference, and fuse it with general preference seamlessly. Although the main argument of this chapter is claimed from the ranking perspective, the implementation is also alongside with negative sampling techniques.

The influence of geographical neighborhood has been demonstrated to be useful in improving preference value prediction (e.g., rating prediction) task. However, few works have exploited it to build a ranking-based model for top-N item recommendations. In this work, we find that each individual's traits are likely to cluster around multiple centers, after a manual inspection of real-world datasets. Hence, this chapter presents a co-pairwise ranking model GeoBPR (geographical Bayesian personalized ranking) based on the observation that users prefer to assign higher ranks to the POIs near previously rated ones. GeoBPR is optimized by location-based negative sampling and stochastic gradient descent approach. Hence, it can learn preference ordering from unobserved training pairs by leveraging location information, and thus can alleviate the sparsity problem of matrix factorization. Evaluation on two publicly available datasets shows that our method performs significantly better than the state-of-the-art techniques for the item recommendation task.

This chapter is mainly based on our previous work "Joint Geo-Spatial Prefer-

ence and Pairwise Ranking for Point-of-Interest Recommendation" published in International Conference on Tools with Artificial Intelligence (ICTAI) 2016 with DOI:10.1109/ICTAI.2016.0018[1]. The paper won the best student paper award in ICTAI2016.

5.1 Introduction

Location-based social networks have emerged as an application to assist users in improving decision-making among a huge volume of point-of-interests (POIs), e.g. bars, stores, and cinemas. Typical location-based websites, such as `yelp.com` and `foursquare.com`, allow users to check-in POIs with mobile devices like smart phones and share tips with online friends (Gao et al., 2015). Yelp, for instance, reaches a monthly average of 83 million unique access via mobile devices, with nearly hundred million reviews by the end of 2015[2]. Since these sites contain a vast amount of valuable information about business popularity and customer preference, making an effective satisfactory decision among POIs has become a serious challenge for individuals. POI recommendation aims to solve such problems by predicting preference scores to unknown POIs and returning the top-N highest ranked POIs for active users according to their previous visits.

At first sight, POI recommendation has no difference from suggesting other types of products e.g. a song or a movie. The key difference, between a POI and other kind of products that have been studied in literature, is that a POI physically exists at a specific spatial area and geocoded by geographical longitude/latitude coordinates (Hu et al., 2014). Moreover, physical interactions between users and POIs play an important role in the check-in decision-making process. Based on this observation, many research techniques e.g., Ye et al. (2011); Yuan et al. (2013); Zhang and Chow (2015) have been proposed to model geographical influence.

However, most recent efforts focus on fitting a check-in frequency based on the user's historical visiting profiles. In particular, different types of contextual information, e.g., geographical coordinates (Cheng et al., 2012), time stamps (Gao et al., 2013), social friends (Ye et al., 2012), categories (Zhang and Chow, 2015)

[1] Copyright: © 2016 IEEE. Reprinted, with permission, from Yuan et al. (2016d)
[2] http://www.yelp.co.uk/press

are incorporated in a single collaborative filtering (CF) model, e.g., matrix factorization (Lian et al., 2014) or a unified framework (Cheng et al., 2012; Yuan et al., 2013; Ye et al., 2011; Zhang and Chow, 2015). Differently, a pairwise based optimization criterion, using the Bayesian theory, has been proposed, which is known as BPR (Rendle et al., 2009b). Prior research has shown that the BPR-based approaches empirically outperform pointwise methods for implicit feedback data (Pan and Chen, 2013; Rendle et al., 2009b) with the same sampling method. However, the BPR-based ranking models do not explicitly exploit geographical influence. Hence, there seems a large marginal space left to improve the performance by extending it for POI recommendation.

We can infer that combining geo-spatial preference and the BPR optimization criterion creates new opportunities for POI recommendation, which has not been investigated yet. Our contributions are as follows: (1) We conduct a manual inspection on real-world datasets, and observe that a user's rating distribution represents a spatial clustering phenomenon. Thus, we presume that unrated POIs surrounded a POI that users prefer are more likely to be assigned higher ranks over the distant unrated ones. In our work, a user's geo-spatial preference is exploited as intermediate feedback, treated as weak preference relative to positive feedback and as strong preference in comparison to other non-positive feedback. (2) We propose a co-pairwise ranking model, which is called geographical Bayesian personalized ranking (GeoBPR) based on the above assumption. In this case, we reformulate the item recommendation problem into a two-level joint pairwise ranking task. To our best knowledge, the reported work is the first to combine BPR optimization criterion with users' geographical preference. (3) We conduct extensive experiments to evaluate the effectiveness of GeoBPR, and the results indicate that our proposed model can significantly outperform an array of counterparts in terms of four popular ranking metrics.

5.2 Related Work for POI recommendation

In this section, we briefly review the recent advances in POI recommendation, particularly those employing geographical influence for POI recommendation.

Recently, a number of valuable works have been presented in the realm of

POI recommendation. Based on the type of additional information involved, POI recommendation algorithms have been classified into four categories (Zhao et al., 2016a), which are (1) pure check-in/rating based POI recommendation approaches (Berjani and Strufe, 2011), (2) social influence enhanced POI recommendation (Cheng et al., 2012; Zhang and Chow, 2015), (3) temporal influence enhanced POI recommendation (Gao et al., 2013; Zhang and Chow, 2015), and (4) geographical influence enhanced POI recommendation (Cheng et al., 2012; Lian et al., 2014; Ye et al., 2011). In particular, in terms of geographical influence enhanced POI recommendation, usual approaches are to assume that users tend to visit nearby POIs and the probability of visiting a new place decreases as the distance increases. For example, Ye et al. (2011) and Yuan et al. (2013) modelled the check-in probability to the distance of the whole visiting history by power-law distribution; Cheng et al. (2012) pointed out that they ignored the geographical cluster phenomenon of users' check-ins, and computing all pairwise distance of the whole visiting history is time-consuming and thus cannot be adapted to large-scale datasets. In contrast, they suggested to model the probability of a user's check-ins as a multi-center Gaussian Model (MGM). Moreover, Zhang and Chow (2015) developed a personalized geographical distribution by applying a Kernel Density Estimation (KDE); Lian et al. (2014) incorporated the spatial clustering phenomenon into matrix factorization to improve recommendation performance. On the other hand, Liu et al. (2013) proposed a two-stage recommendation framework, where at the first stage, users' preference transitions are predicted by a basic matrix factorization (MF) model, and at the second stage, users' preferences for locations in the corresponding categories are inferred by another MF model. In contrast with the aforementioned works, the proposed geographical Bayesian personalized ranking (GeoBPR) is designed by fitting a user's preference rankings for POIs, instead of fitting her check-in counts as traditional factorization methods do. Similarly, (Li et al., 2015a) designed a pairwise ranking model (Rank-GeoFM) based on the Ordered Weighted Pairwise Classification (OWPC) criterion that can incorporate differentcontextual information. The difference is that GeoBPR incorporates geographical preference with BPR criterion, while Rank-GeoFM incorporates geographical preference by prediction function, i.e., a matrix factorization model.

Table 5.1: Basic statistics of Datasets.

DataSets	#Users	#POIs	#Ratings	Density
Phoenix	4510	16402	226351	0.31%
Las Vegas	4470	11376	207649	0.41%
DataSets	Avg.U	N.50	N.100	N.200
Phoenix	50.19	2.51	5.26	12.6
Las Vegas	46.45	5.54	9.44	20.8
DataSets	N.400	N.600	N.1000	N.2000
Phoenix	29.3	46.1	80.26	189.8
Las Vegas	47.6	77.3	146.1	392.3

The "Density" column is the density of each dataset (i.e. Density=#Ratings/(#Users × #Items)). The "Avg.U" column denotes the average number of visited POIs for each user. The "N.k" column refers to the average number of geographical neighbors for a POI at radius k.

It is worth noticing that POI recommendation falls in the category of implicit feedback setting. As a result, most implicit recommendation models can be used for POI recommendation. For example, the original Bayesian personalized ranking (BPR) model based on pairwise preference comparison over observed and unobserved rating pairs. The work in (Rendle et al., 2009a) extended BPR-based matrix factorization with tensor factorization. They further suggested to apply adaptive and context-dependent oversampling to replace the uniform sampling of BPR (Rendle and Freudenthaler, 2014). In (Krohn-Grimberghe et al., 2012; Zhao et al., 2014), BPR criterion was extended by modeling social relations and social preference information. However, since these models do not explicitly leverage the location information, they may lose important location information and thus lead to suboptimal recommendations.

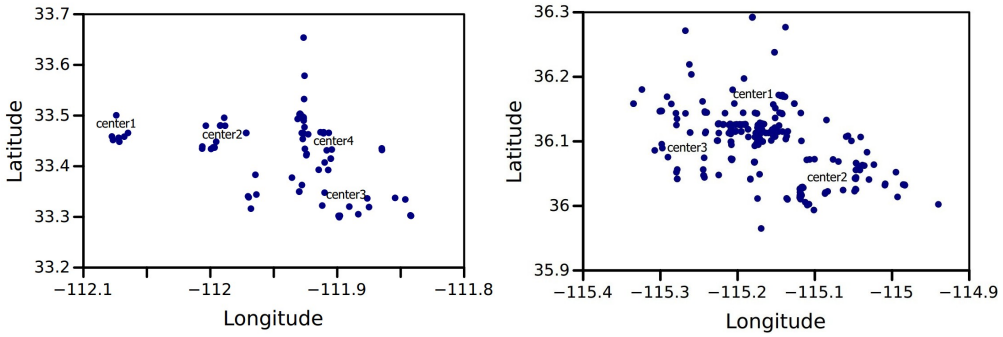
5.3 Geo-spatial Preference Analysis

5.3.1 Data Description

A recently released dataset Yelp[1] is used for data analysis. We extract data from two American cities (Phoenix and Las Vegas) and remove users with less than 20 ratings and POIs with less than 5 ratings to reduce noise data[2], similarly as preprocessed in Cheng et al. (2012); Hu et al. (2014); Liu et al. (2015); Shi et al.

[1] www.yelp.co.uk/dataset_challenge

[2] The cold-start problem is beyond the concern of this work.



(a) Phoenix (b) Las Vegas

Figure 5.1: The overview of a random user's multi-center mobility behaviours on Phoenix and Las Vegas.

(2012b). The basic statistics are shown in Table 5.1.

5.3.2 Motivation

Motivation 1: Our first motivation derives from the Tobler's First Law of Geography, which is "Everything is related to everything else, but near things are more related than distant things" (Tobler, 1970). This implies: (1) a user tends to visit nearby places (Ye et al., 2011); (2) nearby places potentially have some relevance (Hu et al., 2014).

Motivation 2: Cheng et al. (2012) observed that users' check-in traces usually follows a multi-center distribution. Accordingly, they modelled the probability of a user's check-ins on a location as Multi-center Gaussian distribution and then fused the users' geographical preference and latent factor together in a unified framework. Ye et al. (2011) argued that the probability of POI pairs visited by the same user approximately obeyed power-law distribution with distance.

As a result, two main implications can be derived: (1) users usually visit POIs close to their activity centers, such as their homes and offices; (2) users may be interested in exploring POIs near a location they visited before, which have to be clustered together.

Table 5.2: Ratios of $\mathcal{P}^{'}/\mathcal{P}$.

μ	50	100	200	400	600	1000	2000
Phoenix	44.7	28.8	23.3	18.9	13.6	10.2	8.0
Las Vegas	57.5	53.2	19.1	8.0	4.7	4.1	3.2

5.3.3 Proximity Analysis

We proceed to study if the above intuitive phenomena can be observed on our datasets. Figure 5.1 depicts the geographical distributions of POIs rated by two random users. It can be seen that the users' POIs indeed cluster around several spatial areas. That is, the above two implications are likely to hold on the Yelp datasets. Furthermore, we design the following statistical experiments to verify the intuitions.

Exp1: We randomly pick two POIs (l_a, l_b) from a city (e.g. Phoenix) and calculate the distance $d_{(a,b)}$ between l_a and l_b. We repeat the experiment 10000 times in order to yield the probability $\mathcal{P}$ that the distance $d_{(a,b)}$ is less than a threshold μ (e.g. $\mu = 200m$, where m is in meter).

Exp2: We randomly pick a user u from the same city in Exp1 and select two POIs that u has rated before, e.g. $(l_{a'}, l_{b'})$, then calculate the distance $d_{(a',b')}$ between $l_{a'}$ and $l_{b'}$. We repeat the experiment 10000 times to calculate the probability $\mathcal{P}^{'}$ that $d_{(a',b')}$ is less than the same threshold μ.

Table 5.2 shows the ratios of $\mathcal{P}^{'}$ to $\mathcal{P}$, which are indicators to demonstrate the proximity influence of individuals' rating behaviors. As shown, the ratios are much greater than 1 with all thresholds ([50, 2000]), which means $\mathcal{P}^{'}$ is higher than $\mathcal{P}$, in particular, $\mathcal{P}^{'}$ is about 20 times larger than $\mathcal{P}$ when μ is less than $200m$. This implies that users' visiting behaviors are highly affected by spatial distance and that the users' rated POIs are not geographically independent of each other. Second, all the ratios decrease with the increase of μ. This is consistent with intuition since the ratio should be close to 1 if μ is large enough. Moreover, this observation keeps consistent with our experimental results in Section 5.6.2. Unlike previous works, we do not model the distribution of multiple spatial cluster phenomenon directly since it is not proper to assume all users mobility patterns correspond to a prior distribution, e.g. Gaussian (Cheng et al., 2012) or power-law (Ye et al., 2010; Yuan et al., 2013) distribution. Nevertheless, the statistical

analysis implies that an unrated POI surrounded a POI that one prefers is likely to be more appealing (to her) compared with other faraway and unrated places. The physical cost is the main difference that distinguishes POI recommendation from other product recommendations. Thus it becomes feasible to statistically model this intuition by a two-level pairwise preference comparison[1].

Preference rank of a POI one rated >
nearby POIs she unrated >
unrated POIs far away from all rated POIs.

5.4 Preliminaries

First we introduce several concepts used in this chapter and define the research problem of geo-spatial preference enhanced POI recommendation. Then we shortly recapitulate the basic idea of Bayesian Personalized Ranking (BPR). Table 5.3 lists the notations used in this work.

5.4.1 Problem Statement

In the context of typical POI recommendation, let $\mathcal{U} = \{u\}_{u=1}^{M}$ denote the set of all users, and $\mathcal{L} = \{i\}_{i=1}^{N}$ denote the set of all POIs, where M and N represent the number of users and POIs respectively, i.e. $M = |\mathcal{U}|$ & $N = |\mathcal{L}|$. Users' check-in/rating information is commonly expressed via user-POI check-in/rating action matrix $\mathcal{C}$, where each entry c_{ui} is the frequency or a binary value[2] made by u at i. Generally, the matrix $\mathcal{C}$ is extremely sparse (see Table 5.1) since most users only rate a small portion of POIs. In contrast to other recommendation tasks, geographical information is available for each POI, which is usually geocoded by a pair of latitude and longitude.

In our scenario, we define the set of user-POI (u, i) pairs as positive feedback if the rating behavior of u to i is observed, denoted as $\mathcal{L}_u^+ = \{(u, i)\}$. Unlike previous works (e.g. Rendle et al. (2009b)) that defines the set of unobserved pairs $\mathcal{L} \backslash \mathcal{L}_u^+$ as negative feedback, we introduce a new geographical feedback by

[1] We noticed that a two-level assumption has also been applied in Zhao et al. (2014), which however solves the social recommendation problem with model derivation from a different perspective.

[2] For item recommendation task, it is common practice to handle explicit rating values as implicit binary values.

exploiting POI neighborhood information. In particular, assume we observe a POI network $\mathcal{G} = (\mathcal{L}, \mathcal{L})$, where $(i, g) \in \mathcal{G}$ indicates that i and g are geographical neighbors. For each rated POI i, there is a neighbor $g \in \mathcal{L}$, which has not been rated by u. The set of (u, g) pairs is defined as geographical feedback, denoted as $\mathcal{L}_{ui}^{\mathcal{G}} = \{(u, g)\}$. Besides, there is a POI j neither rated by u nor a geographical neighbor of all rated POIs $i \in \mathcal{L}_u^+$. We define the set of (u, j) pairs as negative feedback, denoted as $\mathcal{L}_u^- = \{(u, j)\}$. For example, the circular area in Figure 5.2 represents the range of geographical neighbors. On the left side, (u, i_1) and (u, i_2) are observed rating pairs, i.e. $\mathcal{L}_u^+ = \{(u, i_1), (u, i_2)\}$; (u, g_1) and (u, g_2) represent unobserved pairs, where g_1 and g_2 are neighbors of i_1 and i_2, respectively, i.e. $\mathcal{L}_{ui_1}^{\mathcal{G}} = \{(u, g_1)\}$, $\mathcal{L}_{ui_2}^{\mathcal{G}} = \{(u, g_2)\}$; (u, j) represents the remaining unobserved pairs, i.e. $\mathcal{L}_u^- = \{(u, j)\}$. On the right side of Figure 5.2, g is a common neighbor of i_1 and i_2[1].

The goal of this work is to recommend each user a personalized ranked list of POIs from $\mathcal{L} \backslash \mathcal{L}_u^+$. Motivated by geo-spatial proximity, the key challenge is to learn individuals' implicit preference by integrating positive, geographical and negative feedback.

5.4.2 BPR: Ranking with Implicit Feedback

POIs that a user has never visited are either really unattractive or undiscovered yet potentially appealing (Lian et al., 2014). This is the key challenge of POI recommendation based on implicit feedback. To tackle it, Rendle et al. (Rendle et al., 2009b) proposed a well-known ranking-based optimization criterion Bayesian personalized ranking (BPR) that maximizes a posterior estimation with Bayesian theory. An intuitive assumption is made: **user u prefers item (i.e. POI in our case) i to item j, provided that (u, i) rating pair is observed and (u, j) is unobserved**, defined by:

$$\hat{r}_{uij}(\Theta) := \hat{y}_{ui}(\Theta) > \hat{y}_{uj}(\Theta), i \in \mathcal{L}_u^+, j \in \mathcal{L} \backslash \mathcal{L}_u^+ \tag{5.1}$$

where Θ denotes a set of parameters of a ranking function (i.e. matrix factorization in this work), $\hat{y}_{ui}(\Theta)$ and $\hat{y}_{uj}(\Theta)$ are the predicted score by the ranking function, $\hat{r}_{uij}(\Theta)$ says i is preferred over j by u. Throughout this work, we will

[1] Further details with Figure 5.2 can also be found in Section 6.4.1.

Table 5.3: List of notations.

Symbols	Meanings
$\mathcal{U}$	set of users $\{u_1, u_2,...,u_{\|\mathcal{U}\|}\}$
$\mathcal{L}$	set of POIs $\{i_1, i_2,...,i_{\|\mathcal{L}\|}\}$
$\mathcal{G}$	geographical network
$\mathcal{L}_u^+$	set of (u, i) pairs
$\mathcal{L}_u^-$	set of (u, j) pairs
$\mathcal{L}_{ui}^{\mathcal{G}}$	set of (u, g) pairs
Θ	model parameters
W	users' latent factor matrix
H	POIs' latent factor matrix
b	POI bias
λ, β	regularization parameters
η	learning rate
k	the dimension of latent factors
$\hat{y}$	the ranking score calculated from decomposed models
$\hat{r}$	the ranking relations of two POIs rated by user u
d	the distance between two POIs

write $\hat{r}_{uij}$ for $\hat{r}_{uij}(\Theta)$ to simplify notation, and the same applies to $\hat{r}_{uig}(\Theta)$, $\hat{r}_{ugj}(\Theta)$, $y_{ui}(\Theta)$, $y_{ug}(\Theta)$ and $y_{uj}(\Theta)$. A unique characteristic of BPR is to sort pairwise preference $\hat{y}_{ui}$ and $\hat{y}_{uj}$ instead of regressing a predictor to a numeric value.

5.5 The GeoBPR Model

5.5.1 Model Assumption

The pairwise preference assumption of BPR, holds in practice, empirically produces much better performance than pointwise prediction methods (Rendle et al., 2009b). However, we observe that there are two drawbacks in the BPR assumption for POI recommendation tasks: (1) The BPR algorithm is originally designed for general item recommendations[1], where the structure of geo-spatial preference has not been explicitly considered. Although the factors decomposed from matrix are sementically latent, there is no evidence showing that the latent space has included geographical features. Furthermore, leveraging geographical influence explicitly has been confirmed effectively as in (Cheng et al., 2012; Lian et al., 2014). (2) A large number of unobserved user-POI pairs cannot be employed for learning since BPR treats non-positive pairs equally. Thus we believe there is

[1] An item can be anything, e.g. a book, a song as well as a POI.

much room for improvement by exploiting geographical proximity influence between users and POIs. Specifically, we propose a novel assumption by explicitly modeling the structure of geographical proximity factors.

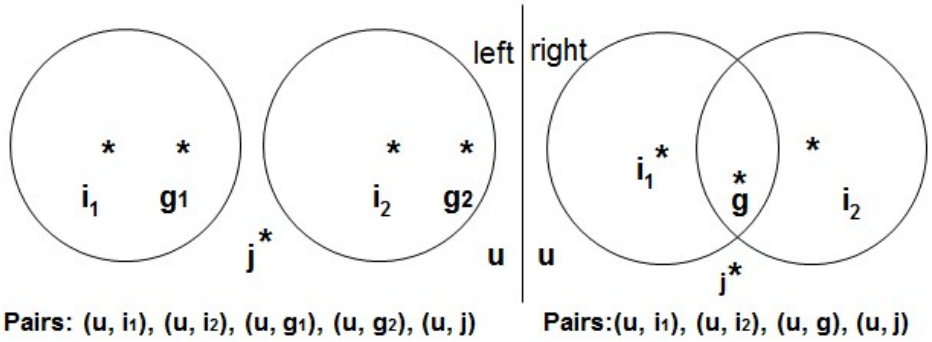


Figure 5.2: Two scenarios of user-POI pairs.

Assumption-a: As stated in section 5.3.3, individuals tend to visit nearby places. Hence, we devise an intermediate process to enhance the BPR assumption: **user u prefers POI i to POI g, provided that (u, i) rating pair is observed and (u, g) is unobserved, where g is one of the geographical neighbors of i; moreover, u prefers g to j, provided that (u, j) is unobserved and j is not a geographical neighbor of all rated POIs**. This assumption can be formulated as follows:

$$\underbrace{\hat{y}_{ui} > \hat{y}_{ug}}_{:=\hat{r}_{uig}} \wedge \underbrace{\hat{y}_{ug} > \hat{y}_{uj}}_{:=\hat{r}_{ugj}}, i \in \mathcal{L}_u^+, g \in \mathcal{L}_{ui}^{\mathcal{G}}, j \in \mathcal{L}_u^- \tag{5.2}$$

It can be seen the preference orders of unobserved pairs, i.e. $(u, g), (u, j)$ are now possible to be compared using our assumption. Thus it seems promising that the sparsity problem is likely to be alleviated. Moreover, based on this assumption and the sound transitivity scheme (Rendle et al., 2009b), it is easy to infer as follows:

$$\underbrace{\hat{y}_{ui} > \hat{y}_{ug}}_{:=\hat{r}_{uig}} \wedge \underbrace{\hat{y}_{ui} > \hat{y}_{uj}}_{:=\hat{r}_{uij}} \tag{5.3}$$

We can see the assumption based on Eq.(5.3) reduces to that of BPR (see Eq.(5.1)). In other words, the new assumption leads to a more accurate interpretation than typical BPR assumption. Furthermore, the proposed assumption

balances the contribution between geographical preference and latent factors[1]. Note that we cannot infer any preference relation from these pairs: $(\hat{y}_{ui_1}, \hat{y}_{ui_2})$, $(\hat{y}_{ui_1}, \hat{y}_{ug_2})$, $(\hat{y}_{ui_2}, \hat{y}_{ug_1})$, $(\hat{y}_{ug_1}, \hat{y}_{ug_2})$) (see Figure 5.2).

Assumption-b: We are also interested in investigating an opposite assumption since unvisited POIs near a frequently visited POI are likely to be unattractive. This is because the user is likely to know about these POIs since they are close to her frequently visited ones, yet she has never chosen to patronize them before. This might be a signal that she dislikes them. In other words, geographical neighbors should be treated more negatively than other unvisited ones. Visiting frequency here is employed as the confidence of a user's preference. However, on the Yelp datasets, each POI has at most one rating by each user. Intuitively, this assumption may not hold without frequency information. For the sake of completeness of this work, we also verify the effectiveness of this assumption, formulated as follows:

$$\underbrace{\hat{y}_{ui} > \hat{y}_{uj}}_{:=\hat{r}_{uij}} \wedge \underbrace{\hat{y}_{uj} > \hat{y}_{ug}}_{:=\hat{r}_{ujg}}, i \in \mathcal{L}_u^+, g \in \mathcal{L}_{ui}^{\mathcal{G}}, j \in \mathcal{L}_u^- \tag{5.4}$$

Due to the space limitations, we merely elaborate the derivation process of our approach with assumption-a and report the final performance of the two assumptions in section 5.

5.5.2 Model Derivation

Based on the above assumptions, we employ a maximum posterior estimator to find the best ranking for a specific user u:

$$\arg\max_{\Theta} \mathcal{P}(\Theta| >_u) \tag{5.5}$$

where Θ represents a set of model parameters as mentioned before, $>_u$ is the total order, which represents the desired but latent preference structure for user u. According to Bayesian theory, the $\mathcal{P}(\Theta| >_u)$ can be inferred as:

$$\mathcal{P}(\Theta| >_u) \propto \mathcal{P}(>_u |\Theta)\mathcal{P}(\Theta) \tag{5.6}$$

where $P(>_u |\Theta)$ is the likelihood function and $\mathcal{P}(\Theta)$ is the prior distribution of parameters Θ. Here three intuitive assumptions are made: (1) Each user's rating actions are independent of every other user. (2) The preference ordering

[1] $\hat{y}$ is usually computed by a latent factor model, i.e. matrix factorization in this work.

of each triple of items (i, g, j) for a specific user is independent of the ordering of every other triple. (3) The preference ordering of (i, g) pair for a specific user is independent of the ordering of (g, j) one. Based on the assumptions, Bernoulli distribution over the binary random variable can be used to estimate the likelihood function as follows:

$$\prod_{u\in\mathcal{U}} \mathcal{P}(>_u|\Theta) = \prod_{(u,i,g,j)\in\mathcal{U}\times\mathcal{L}\times\mathcal{L}\times\mathcal{L}} \mathcal{P}\left(\hat{y}_{ui} > \hat{y}_{ug} \wedge \hat{y}_{ug} > \hat{y}_{uj}|\Theta\right)^{\delta((u,i,g,j)\in D_s)} \cdot \left(1 - \mathcal{P}(\hat{y}_{ui} > \hat{y}_{ug} \wedge \hat{y}_{ug} > \hat{y}_{uj}|\Theta)\right)^{\delta((u,i,g,j)\notin D_s)} \tag{5.7}$$

D_s is a poset of $>_u$, which expresses the fact that that user u is assumed to prefer i over g, and prefer g over j, i.e., $D_s = \left\{(u, i, g, j)|i \in \mathcal{L}_u^+ \wedge g \in \mathcal{L}_{ui}^{\mathcal{G}} \wedge j \in \mathcal{L}_u^-\right\}$. $\delta(x)$ is a binary indicator with $\delta(x) = 1$ if x is true and $\delta(x) = 0$, otherwise. Due to the totality and antisymmetry (Rendle et al., 2009b) of a pairwise ordering scheme, Eq.(5.7) can be simplified to:

$$\prod_{u\in\mathcal{U}} \mathcal{P}(>_u|\Theta) = \prod_{u\in\mathcal{U}, i\in\mathcal{L}_u^+, g\in\mathcal{L}_{ui}^{\mathcal{G}}} \mathcal{P}(\hat{y}_{ui} > \hat{y}_{ug}|\Theta) \prod_{u\in\mathcal{U}, g\in\mathcal{L}_{ui}^{\mathcal{G}}, j\in\mathcal{L}_u^-} \mathcal{P}(\hat{y}_{ug} > \hat{y}_{uj}|\Theta) \tag{5.8}$$

We employ a differential function, e.g., $\sigma(x) = \frac{1}{1+e^{-x}}$, to approximate the probability $\mathcal{P}(.)$ and map the value to probability range (0, 1). Unlike previous works, e.g., Rendle et al. (2009b), which assign an equal weight to each training pair, we design two trivial weight functions, w_{ig} and w_{gj}, to relax the pairwise preference assumption between $\hat{y}_{ui}$ and $\hat{y}_{ug}$, $\hat{y}_{ug}$ and $\hat{y}_{uj}$. Specifically, the two estimators can be derived as:

$$\begin{aligned} \mathcal{P}(\hat{y}_{ui} > \hat{y}_{ug}|\Theta) &= \frac{1}{1 + e^{-w_{ig}(\hat{y}_{ui} - \hat{y}_{ug})}} \\ \mathcal{P}(\hat{y}_{ug} > \hat{y}_{uj}|\Theta) &= \frac{1}{1 + e^{-w_{gj}(\hat{y}_{ug} - \hat{y}_{uj})}} \\ w_{ig} = \frac{1}{1 + n_{ig}}, w_{gj} &= \frac{1}{1 + (1 + d)^{-1}} \end{aligned} \tag{5.9}$$

where w_{ig} is to control the contribution of sampled training pair (u, i) and (u, g) to the objective function, n_{ig} is the number of rated POIs that are geographical neighbors of POI g. w_{ig} equals 1 if no other rated POI shares g as a geographical neighbor, and the value decreases if g is a public geographical neighbor. The reason behind is that the above assumption of pairwise preference may not always hold in real applications. For example, POI g may be a geographical neighbor of more than one rated POIs (see right side of Figure 5.2). In this case, a user u may

potentially prefer an POI g to POI i because g is close to the activity center of u. Similarly, w_{gj} is used to control the contribution of the training pair (u, g) and (u, j), d is distance[1] between POI g and POI j. This is obvious as the preference ordering is correlated with the distance value between g and j, e.g. the preference assumption between (u, g) and (u, j) pairs may not hold if d is small. With the settings of w_{ig} and w_{gj}, the contribution of geographical preference works more reasonably.

Regarding prior density $\mathcal{P}(\Theta)$, it is common practice to design a Gaussian distribution with zero mean and model specific variance-covariance matrix $\lambda_\Theta I$,

$$\mathcal{P}(\Theta) \sim \mathcal{N}(0, \lambda_\Theta I) \tag{5.10}$$

Finally, we reach the objective loss function of our GeoBPR:

$$\begin{aligned} GeoBPR :=& argmax_\Theta \mathcal{P}(\Theta|\mathcal{D}_s) := argmin_\Theta (\lambda_\Theta ||\Theta||^2 \\ &- \sum_{u \in \mathcal{U}, i \in \mathcal{L}_u^+, g \in \mathcal{L}_{ui}^{\mathcal{G}}} \ln \sigma(w_{ig}(\hat{y}_{ui} - \hat{y}_{ug})) \\ &- \sum_{u \in \mathcal{U}, g \in \mathcal{L}_{ui}^{\mathcal{G}}, j \in \mathcal{L}_u^-} \ln \sigma(w_{gh}(\hat{y}_{ug} - \hat{y}_{uj}))) \end{aligned} \tag{5.11}$$

The prediction function $\hat{y}$ is modelled by matrix factorization, which is known to effectively discover the underlying interactions between users and items.

$$\hat{y}_{ui} = W_u \cdot H_i^T + b_i = \sum_{f=1}^{k} w_{u,f} \times h_{i,f} + b_i \tag{5.12}$$

where W_u and H_i^T represent latent factors of user u and POI i resp., b_i is the bias term of i[2]., i.e. $\Theta = \{W \in R^{\mathcal{U} \times k}, H \in R^{\mathcal{L} \times k}, b \in R^{\mathcal{L}}\}$. The similar ranking functions apply to $\hat{y}_{ug}$ and $\hat{y}_{uj}$.

According to Eq.(5.11) & Eq.(5.12), we observe that GeoBPR models the user's preference rankings by taking into account two types of factors: (1) due to the spatial proximity of (i, g) pair, the difference of $(\hat{y}_{ui}, \hat{y}_{ug})$ models the preference relations mostly based on general latent features[3], such as users' latent factors (i.e. the taste of the user) and POI latent factors (e.g. categories, price, reputation,

[1] The metric unit of d is 1000 meters.

[2] Note that the user bias term vananishes for predicting rankings and for optimization as the pairwse comparison is based on one user level.

[3] Despite the success of latent factorization models, there is no literature to uncover the specific structure of latent factors, which is also beyond the scope of this thesis.

etc.); (2) by explicitly modeling the difference of ($\hat{y}_{ug}$, $\hat{y}_{uj}$), the factorization model is likely to learn more about the structure of geo-spatial preference. That is, by injecting geo-spatial preference and leveraging latent factor models (e.g. matrix factorization), GeoBPR balances the proximity influence and the latent features to learn the best personalized rankings.

5.5.3 Model Learning and Sampling

Algorithm 6 GeoBPR Learning

Input: $\mathcal{D}_s$, $\mathcal{G}(\mathcal{L}, \mathcal{L})$
Output: model parameters Θ
Initialize Θ with Normal distribution $\mathcal{N}(0, 0.1)$
for $u \in \mathcal{U}$ **do**
    Calculate $\mathcal{L}_u^+$, $\mathcal{L}_{ui}^{\mathcal{G}}$, $\mathcal{L}_u^-$
end for
repeat
    for $u \in \mathcal{U}$ **do**
        Uniformly draw (i, g, j) from $\mathcal{L}_u^+$, $\mathcal{L}_{ui}^{\mathcal{G}}$, $\mathcal{L}_u^-$
        Calculate c_{ig}, c_{gj}, i.e.
        $c_{ig} = \frac{1}{1+e^{\frac{\hat{y}_{ui}-\hat{y}_{ug}}{1+n_{ig}}}} \cdot w_{ig}, c_{gj} = \frac{1}{1+e^{\frac{\hat{y}_{ug}-\hat{y}_{uj}}{1+(1+d)^{-1}}}} \cdot w_{gj}$
        $W_u \leftarrow W_u + \eta(c_{ig}(H_i - H_g) + c_{gj}(H_g - H_j) - \lambda_u W_u)$
        $H_i \leftarrow H_i + \eta(c_{ig} W_u - \lambda_i H_i)$
        $H_g \leftarrow H_g + \eta(-c_{ig} W_u + c_{gj} W_u - \lambda_g H_g)$
        $b_i \leftarrow b_i + \eta(c_{ig} - \beta_i b_i)$
        $b_g \leftarrow b_g + \eta(-c_{ig} + c_{gj} - \beta_g b_g)$
        $b_j \leftarrow b_j + \eta(-c_{gj} - \beta_j b_j)$
    end for
until convergence **return** Θ

Since Eq.(5.11) is differentiable, we adopt the widely used stochastic gradient descent (SGD) and negative sampling for optimization, such as LambdaFM (Yuan et al., 2016b) and BoostFM (Yuan et al., 2017). Different from them, the sampled negative POIs are from two sources: neighbor POIs and distant POIs. Specifically, for each user u, we randomly select (i, g, j) triples from positive, geographical and negative feedback, and then iteratively update parameters Θ. The update equations are given in Algorithm 6. Intuitively, by explicitly leveraging

the location information and sampling more informative negative examples, the performance of BPR may have a chance to be improved. In our experiments, we empirically verify the performance of our GeoBPR by comparing to the original BPR. Regarding the computational complexity, we can see that each update rule is $O(k)$, where k is the number of latent dimensions. The total complexity is $O(T|\mathcal{U}|k)$, where T is the number of iterations. For predicting a user's preference on a POI, the complexity is linear $O(k)$. Both learning and predicting processes do not increase the time complexity in contrast with BPR.

5.6 Experiments

5.6.1 Experimental Setup

Yelp datasets described in section 5.3.1 are used for evaluation. All experiments are conducted by using the 5-fold cross-validation. That is, we randomly preserve 80% of the dataset as a training set, and leave the remaining portions as a test set. We report the mean of the five runs as final performance.

5.6.1.1 Baseline Methods

We compare GeoBPR[1] with an bunch of state-of-the-art baselines, including Most Popular (MP), User-based Collaborative Filtering (UCF), MFM, BPR and NBPR. MP, UCF and BPR have been described in Chapter 3.

- **MFM**: The basic idea of MFM is to fuse Multi-center features (Cheng et al., 2012) with Factorization Machines (FM) (Rendle, 2010, 2012). Following Cheng et al. (2012), we produce several clusters based on a user's previously visited POIs and get the average coordinate of each cluster as centroid. The distance between a candidate POI and each centroid is calculated as features. Then we apply FM to model users' latent preference and geo-spatial influence.

- **NBPR**: Inspired by Hu et al. (2014), we implement an intuitive baseline that fuses geographical neighborhood with matrix factorization, and then adopts BPR criterion for learning. The major difference between GeoBPR

[1] GeoBPR is short for GeoBPR with assumption-a if no special instructions.

Table 5.4: Performance comparison. PX and LV denotes Phoenix and Las Vegas respectively. GBPRa and GBPRb denote GeoBPR with assumption a and b respectively.

	Metrics	MP	UCF	MFM	BPR	NBPR	GBPRb	GBPRa
PX	MAP	0.0152	0.0187	0.0153	0.0310	0.0316	0.0095	**0.0335**
	MRR	0.0705	0.0939	0.1129	0.1244	0.1286	0.0507	**0.1406**
LV	MAP	0.0259	0.0372	0.0403	0.0419	0.0426	0.0167	**0.0462**
	MRR	0.0940	0.1422	0.1385	0.1467	0.1484	0.0702	**0.1656**

and NBPR is that GeoBPR embeds the geographical proximity influence into the pairwise rank assumption (i.e. objective function), whereas the NBPR integrates the geographical neighborhood with a more complicated prediction function.

5.6.1.2 Parameter Settings

GeoBPR has several critical hyperparameters to be tuned:

- Learning rate η: To conduct a fair comparison, we apply the 5-fold cross-validation to find the best η for BPR ($\eta = 0.05$), and then employ the same value for GeoBPR. For MFM and NBPR, we apply the same procedure to tune η ($\eta = 0.005$).

- Factorization dimension k: We fix $k = 30$ for all models based on matrix factorization. The effect of k value will be detailed later.

- Regularization λ, β: In this work, regularization parameters are grouped as GeoBPR has several parameters: λ_u represents the regularization parameter of W_u; λ_π represents the regularization parameters of H_i, H_g, H_j (i.e. λ_i, λ_g, λ_j resp.); β_π represents regularization parameters of b_i, b_g, b_j (i.e. β_i, β_g, β_j resp.). On Phoenix dataset, $\lambda_u = 0.03$, $\lambda_\pi = 0.03$, $\beta_\pi = 0.05$; on Las Vegas dataset, $\lambda_u = 0.08$, $\lambda_\pi = 0.02$, $\beta_\pi = 0.05$.

- Initialization Θ: It is common practice to sample a zero-mean normal distribution with a small standard deviation σ. We set $\sigma = 0.1$.

5.6.1.3 Evaluation Metrics

In order to measure the quality of top-N recommendation task, we choose four standard evaluation metrics, namely Precision@N and Recall@N (denoted by Pre@N and Rec@N respectively) (Li et al., 2015a), Mean Average Precision (MAP) (Rendle and Freudenthaler, 2014; Liu et al., 2013) and Mean Reciprocal Rank (MRR) (Shi et al., 2012b). For each metric, we first calculate the performance of each user from the test data, and then obtain the average performance over all users. The higher values MAP, MRR, Pre@N and Rec@N, the better recommendation performance. Due to the space limitations, we leave out more detailed descriptions.

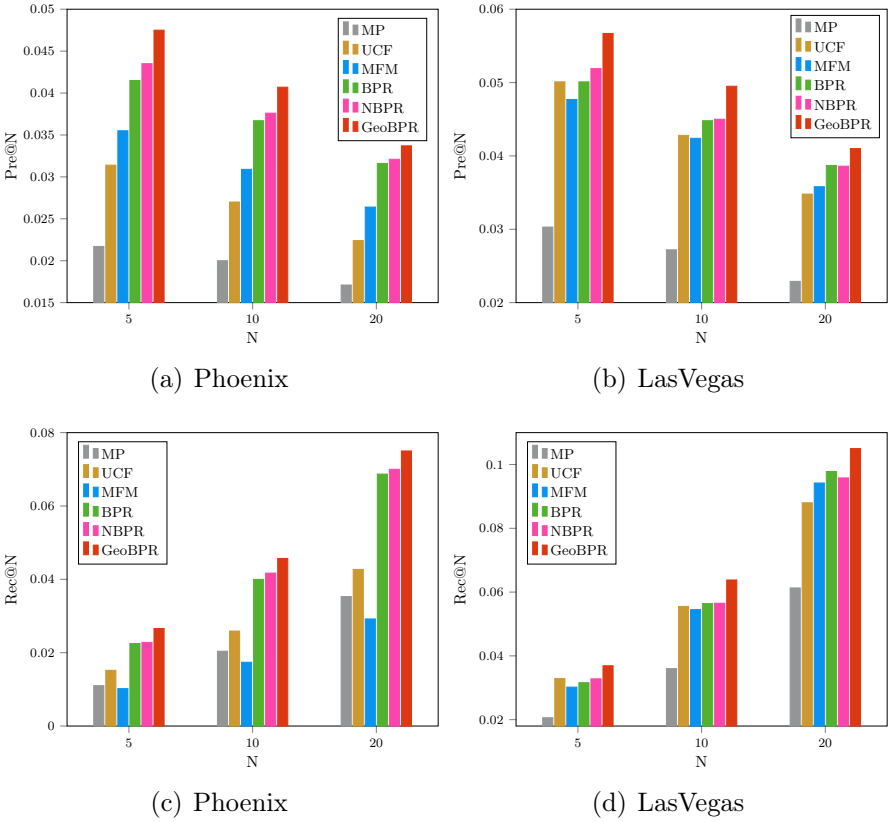
5.6.2 Experimental Results

5.6.2.1 Summary of Experimental Results

Table 5.4 and Figure 5.3 present the experimental results of each algorithm in terms of the four ranking metrics.

We highlight the results of BPR and GeoBPR in boldface for comparison in Table 5.4[1]. The percentage in 'Improve' column represents the accuracy improvement of GeoBPR relative to BPR. As shown, BPR, NBPR and GeoBPR models perform much better than MP, MFM and UCF, which demonstrates the effectiveness of pairwise preference assumptions. Our approach outperforms the other baseline methods in terms of all the metrics on both datasets. In particular, our GeoBPR model achieves about 10% significant improvement compared to the BPR model in terms of MAP and MRR. The main reason is that BPR only learns one ranking order between observed and unobserved POI pairs, i.e. (i, j). While our GeoBPR model learns two orders: rated POI i and nearby POI g which is unrated, i.e. $\hat{r}_{uig}$; both unrated POIs g and j, but j is distant from all rated POIs, i.e. $\hat{r}_{ugj}$. Intuitively, the assumption $\hat{r}_{uig}$ holds more accurately than $\hat{r}_{uij}$ in real scenarios; in addition, sparsity problem seems to be alleviated by the additional assumption $\hat{r}_{ugj}$. We can thus see that the assumption of GeoBPR by injecting geo-spatial preference is indeed more effective than that of simple pairwise preference assumed in BPR. Interestingly, one may observe that NBPR

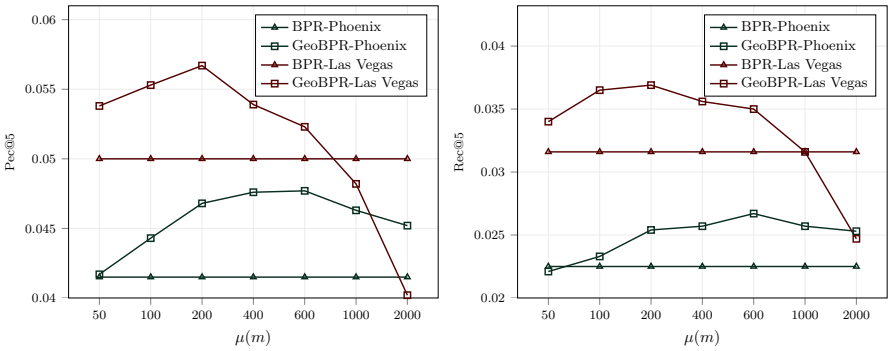
[1] Note that the absolute precision and recall are usually very low in the POI recommendation task due to the huge sparsity of data, which is consistent with previous literature (Zhao et al., 2016b).



(a) Phoenix (b) LasVegas

(c) Phoenix (d) LasVegas

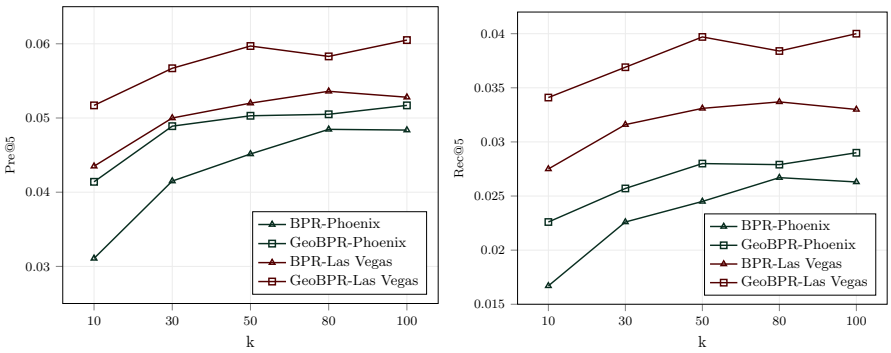
Figure 5.3: Performance comparison with respect to top-N values in terms of Pre@N and Rec@N.



(a) Pre@5 (b) Rec@5

Figure 5.4: Performance comparison with different μ.

does not perform much better than BPR by modeling a new prediction function, i.e. fusing geographical neighborhood with matrix factorization. Our results po-



(a) Pre@5 (b) Rec@5

Figure 5.5: Performance comparison with different k.

tential imply that algorithms optimized for rating prediction do not translate into accuracy improvements in terms of top-N item recommendation. In addition, the results keep consistent with the findings in (Kothari and Wiraatmadja).

5.6.2.2 Impact of Neighborhood

Table 5.4 shows the prediction quality of GeoBPR with two opposite assumptions, i.e. assumption-a and assumption-b (denoted by GeoBPRa, GeoBPRb resp.). As stated in section 5.5.1, assumption-b intuitively cannot hold due to the lack of frequency information, we can see that GeoBPRb performs the worst over other approaches. The results, i.e. Pre@5 and Rec@5[1], are depicted in Figure 5.4. We observe the general trends are, both metrics increase with the increasing of threshold μ, when arriving at a certain threshold, the performance starts decreasing with a larger μ. The reason is that the size of nearby POIs ($g \in \mathcal{L}_{ui}^{\mathcal{G}}$) is not as large as required for training the model[2] when μ is small (e.g. $\mu \in [50, 200]$) (see Algorithm 6 and Table 5.1). Once the number of training samples is large enough, the performance of GeoBPR keeps consistent with the ratio value ($\mathcal{P}'/\mathcal{P}$), i.e. the larger ratio the training samples have, the better recommendation quality GeoBPR achieves. For example, the ratio on Las Vegas dataset become smaller when μ is larger than 600m, and accordingly the recommendation accuracy degrades rapidly when μ increases. Furthermore, we see GeoBPR always

[1] The performance on other metrics follows similar trends.

[2] Once again, we emphasize that the ranking comparison of (i, g) pairs models general latent factors, while the comparsion of (g, j) explicitly models geo-spatial preference.

achieves better performance than BPR on Phoenix dataset when μ in $[100, 2000]$; on both datasets, it outperform BPR when μ in [100, 600].

With respect to the effect of w_{ig} and w_{gj}, we compare GeoBPR with a variant without using the two weights. We observe that for both datasets, it improves the performance by the percent in the range of [2.66% , 4.02%], demonstrating the effectiveness by relaxing the pairwise assumptions. We see the contribution of geo-spatial preference is clear and the results match our data analysis.

5.6.2.3 Impact of Factorization Dimensions

In this work, we apply a matrix factorization (MF) as the scoring function for GeoBPR (see section 5.5.2). Thus it is important to investigate the impact of factorization dimension k to the prediction quality. As shown in Figure 5.5, the performance of BPR and GeoBPR steadily rise with the increasing number of dimensions, which keeps consistent with previous works, e.g., Rendle et al. (2009b). Furthermore, GeoBPR consistently outperforms BPR with the same number of dimensions on both datasets; in particular, the performance of GeoBPR in 30 dimensions is comparable with that of BPR in 100 dimensions.

5.7 Chapter Summary

In this chapter, we explored to leverage geographical influence to improve personalized POI recommendation. We first conducted proximity data analysis on two real-world datasets extracted from the Yelp Datasets and observed that a user's rated POIs tend to cluster together on the map. We argued that users are likely to visit nearby places. Then, we proposed a co-pairwise ranking model (GeoBPR) by injecting the geo-spatial preference. The intermediate proximity preference introduced by geographical feedback leads to a more accurate interpretation than original BPR in the setting of POI recommendation, and makes the preference ordering of unobserved user-POI pairs possible to be inferred. Due to the optimal balance of geographical preference and latent factors, GeoBPR outperformed other state-of-the-art factorization models. Both the theoretical and empirical results indicated that GeoBPR was a good choice for personalized POI recommendation task.

Although GeoBPR is derived from the pairwise ranking perspective its performance improvements also benefit from the negative sampling strategy to some extent since the unobserved POIs that are close to the user's activity center may contain more important information compared with distant ones. The pairwise comparison between the observed POI and nearby unobserved POIs is more meaningful than that with distant unobserved POIs. Hence, from this perspective, the work of GeoBPR can also support our thesis **statement (1)** and **statement (2)**, although its main contribution does not concentrate on developing negative sampling methods.

Part III
Batch Gradient with All Samples

Stochastic gradient descent (SGD) with negative item sampling is the most prevalent optimization technique for recommendation with large-scale implicit feedback data, as discussed in the last three chapters. However, the training time and effects of SGD depend largely on the sampling distribution of negative examples. Sampling a small fraction of unobserved examples as negative for training may ignore other useful examples, or lead to insufficient training of them. Moreover, the high variance of stochastic samples tends to cause the learning fluctuate heavily in the vicinity of a global/local optimum. To address the issue, we derive a fast batch gradient decent (f_{BGD}) for learning generic recommendation models from whole implicit data. Experiments on real-world data sets demonstrate the efficacy of f_{BGD}, which not only achieves a $400-1,000$x speed-up over BGD, but also significantly outperforms state-of-the-art SGD baselines.

This part studies systematically the impact of negative sampling based method and empirically compares it with whole data based method. It provides supports for all the statements in Chapter 1.

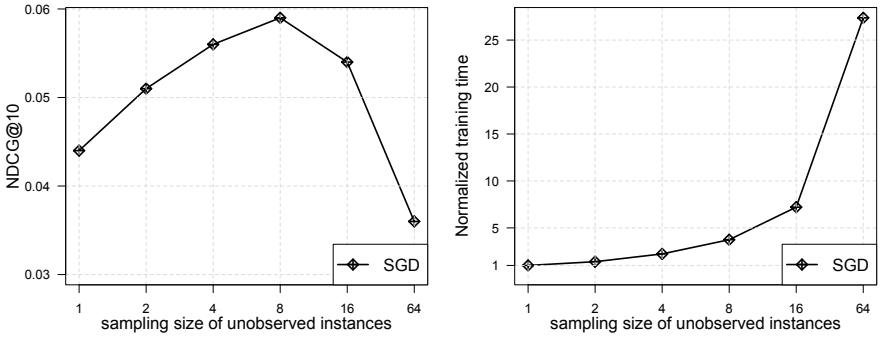
Chapter 6

Fast Batch Gradient Descent

In Part II, we have shown that stochastic gradient descent (SGD) with negative item sampling is the most prevalent optimization technique for recommendation with large-scale implicit feedback data. However, the training time and effects of SGD depend largely on the sampling distribution of negative examples. To address the issue, we derive a fast batch gradient descent f_{BGD} for learning generic recommendation models from whole implicit data. The key design is based on a series of optimized mathematical derivation of the objective function. In addition, we demonstrate that the standard batch gradient method suffers from gradient exploding and vanishing issues when learning complex recommender models due to large batched summation of sparse features. Driven by a theoretical analysis for this potential cause, an easy solution arises in a natural way. Experiments on real-world data sets demonstrate the efficacy of f_{BGD}, which not only achieves a $400-1,000$x speed-up over BGD, but also significantly outperforms state-of-the-art SGD baselines.

This chapter is mainly based on our recent work "f_{BGD}: Learning Embeddings From Positive Unlabeled Data with BGD" published in The Conference on Uncertainty in Artificial Intelligence (UAI) 2018[1]. A special case paper "Batch IS NOT Heavy: Learning Word Representations From All Samples" for the word embedding task was published in Annual Meeting of the Association for Computational Linguistics (ACL) 2018[2].

[1]Copyright: © 2018 UAI. Reprinted, with permission, from Yuan et al. (2018b)
[2]Copyright: © 2018 ACL. Reprinted, with permission, from Xin et al. (2018)



(a) Accuracy (b) Training time

Figure 6.1: Impact of negative item sampling on SGD on the Last.fm data set. The x-axis is the number of sampled negative items for each positive one. Better accuracy (a) can be obtained by sampling more negative instances at the expense of longer training times (b). More details on the experimental settings are given in Section 6.6.

6.1 Introduction

Learning a recommendation model from implicit feedback is challenging, primarily due to the lack of negative feedback. While a common strategy is to treat the unobserved feedback (i.e. missing data) as a source of negative examples (He et al., 2017), the technical difficulties cannot be overlooked: (1) the ratio of positive to negative feedback in practice is highly imbalanced (Pan et al., 2008; Hu et al., 2008), and (2) learning through all unobserved feedback (which is usually billion to trillion level) is computationally expensive (Yuan et al., 2016b).

The SGD with negative item sampling is the most widely used optimization method to solve the large-scale implicit feedback recommendation problem (Pan et al., 2008; Rendle and Freudenthaler, 2014; Zhang et al., 2013; Liu et al.; Mikolov et al., 2013; Yuan et al., 2016b), particularly for complex embedding (or factorization) models such as Factorization Machines (FM) (Rendle, 2012; Hong et al., 2013; Yuan et al., 2017) and SVDfeature (Chen et al., 2012). Despite its success, it is known that SGD may perform frequent gradient updates with a high variance, which can cause the objective function to fluctuate dramatically near the global/local optimum (Ruder, 2016). More importantly, the training time and prediction accuracy are largely determined by the sampling size and distribution of negative examples, as shown in Figure 6.1. In fact, sampling methods are

biased (Blanc and Rendle, 2017), i.e., it does not converge to the same loss with full-sample — regardless how many update steps are taken. This is our main motivation in this work. It is known that batch gradient descent (BGD) computes the gradient on the whole training data for updating a model parameter. As such, the learning process can converge to a better optimum (Pearlmutter, 1992). Unfortunately, although BGD has a more stable convergence, the low efficiency caused by the full batch process makes it almost non-scalable to data with a large number of users and items, and thus is much less explored in literature.

In this chapter, the main contribution is in the deriving of an efficient BGD learning algorithm f_{BGD} — which addresses the computational difficulty of the standard BGD — for learning generic recommender models from implicit feedback. Unlike SGD based models that rely on sampling negative feedback data, f_{BGD} takes the whole implicit data as input, but only has a comparable complexity with negative sampling based SGD. This is realised by a series of rigorous mathematical reasonings in theory and the clever use of memoization strategies. The second contribution is that we provide insight as to why the BGD is prone to the gradient exploding and vanishing problem when the embedding model incorporates features beyond user ID and item ID. An intuitive solution can be applied once we understand the cause. To our knowledge, neither of the two contributions has been made before. In our experiments, we perform a rigorous study and comparison between negative sampling based stochastic gradient models and whole data based batch gradient model. Our findings demonstrate both the significantly improved recommendation accuracy and superior scalability of f_{BGD}.

6.2 Related Work

Gradient methods are one of the most popular algorithms to perform optimization in the practice of machine learning. They have also been widely used for training Collaborative Filtering (CF) models, and have almost dominated the optimization field for embedding models (Koren et al., 2009; Rendle et al., 2009b; Zhao et al., 2014; Mikolov et al., 2013; Hong et al., 2013; Chen et al., 2012; Rendle, 2012; Li et al., 2015b; Yuan et al., 2016b). So far the most commonly used gradient

optimization method is SGD or a compromise MGD (mini-batch gradient descent) (Mikolov et al., 2013; Wang et al., 2017a), which attempts to approximate the true gradient by a single or a mini-batch of instances with sampling techniques. However, the balance between computing the expensive true gradient based on the whole batch and the immediate gradient based on a single instance could easily result in suboptimal performance. More importantly, on large data the sampling size and quality for SGD/MGD also significantly affect the convergence rate and prediction accuracy. In particular for recommender systems, it is non-trivial to sample from large and highly imbalanced implicit feedback data. Most works deal with this issue by proposing a certain trade-off between efficiency and accuracy. For example, various sampling methods have been proposed in recent literature (Pan et al., 2008; Rendle and Schmidt-Thieme, 2010; Weston et al., 2012; Zhao et al., 2014; Zhang et al., 2013; Mikolov et al., 2013; Hong et al., 2013; Yuan et al., 2016b, 2017; Wang et al., 2017a). The basic idea behind this is to select the most informative instances for an SGD/MGD trainer which, however, easily leads to bias itself. Moreover, all aforementioned works either expose efficiency issues with a dynamic sampler (Weston et al., 2012, 2013; Zhang et al., 2013; Hong et al., 2013; Li et al., 2015b; Yuan et al., 2016b) or result in suboptimal training instances with a uniform (Rendle et al., 2009b; Rendle and Schmidt-Thieme, 2010; Liu et al.) or static (defined before optimization) sampler (Pan et al., 2008; Zhao et al., 2014; Mikolov et al., 2013; Hidasi et al., 2015; Yuan et al., 2017) in practice.

It is worth mentioning that the f_{BGD} method is inspired from the extensive empirical studies on previous works (He et al., 2016c; Bayer et al., 2017). The main difference is that He et al. (2016c) worked on the simple matrix factorization model, which cannot be used to incorporate other features, such as contextual variables associated with each observed example. While the alternating least squares (ALS) method proposed in Bayer et al. (2017) can be applied to any *k-separable*[1] mode, it requires to estimate the second-order derivatives to apply the Newton update and only supports a constant weight on unobserved examples; moreover, our empirical evidence shows that training with Newton update is very sensitive to initialization point and the regularization term, and sometimes

[1] In essence, the concept of k-separable is to describe a model with a dot product structure.

is highly unstable due to some gradient issues, especially for embedding models (e.g., FM and SVDFeature) with many input features. By contrast, our f_{BGD} only employs the first-order derivatives and supports fine-grained weighting scheme for unobserved examples.

6.3 Preliminaries

In this section, we discuss in depth the standard batch gradient optimizer for learning embedding models from implicit feedback, and highlight the inefficiency issues of the standard form of BGD. Note that the formulation of the embedding models is slightly different from that in LambdaFM (Chapter 3) and BoostFM (Chapter 4), which are based on the optimization of SGD.

6.3.1 The Generic Embedding Model

Let U denote a set of user-fields and I a set of item-fields. In implicit feedback data, there are a set of observed (user-field, item-field) actions S and unobserved actions $(U \times I) \backslash S$. We define $\mathbf{x}_u^U$ as a feature vector that describes a user-field u with pU real-valued features, and $\mathbf{x}_i^I \in \mathbb{R}^{pI}$ as the item feature vector of i with pI features. We can then store the feature data for all users and items in matrix $\mathbf{X}^U \in \mathbb{R}^{|U| \times pU}$ and $\mathbf{X}^I \in \mathbb{R}^{|I| \times pI}$ respectively, where $|U|$ and $|I|$ denote the number of distinct user- and item-fields, respectively. See Figure 6.2 for an illustration. Accordingly, any input feature vector $\mathbf{x}$ in training data $\mathbf{X}^{\mathrm{S}} \in \mathbb{R}^{|S| \times n}$ $(m = pU + pI)$ can be represented as $\mathbf{x} = (\mathbf{x}_u^U, \mathbf{x}_i^I)$.

Here, we first propose to optimize a general feature-based embedding model (Chen et al., 2012) that is given below, which can be used in various recommendation tasks, such as collaborative filtering (CF) and context/content-based recommendations. Then we introduce the generality of f_{BGD} by identifying a similar structure.

$$\hat{y}_{ui}(\mathbf{x}) = w_0 + \sum_{d=1}^{n} w_d x_d + \sum_{f=1}^{k} (\sum_{j=1}^{pU} x_{u,j}^U v_{j,f}^U)(\sum_{j=1}^{pI} x_{i,j}^I v_{j,f}^I) \tag{6.1}$$

where w_d is the d-th element in $\mathbf{w} \in \mathbb{R}^n$, and $v_{j,f}^U$ is the element of the j-th row and f-th column in $\mathbf{V}^U \in \mathbb{R}^{pU \times k}$, similarly for $v_{j,f}^I$, $x_{u,j}^U$, and $x_{i,j}^I$. w_0, w_d, $v_{d,f}$ are the model parameters to be estimated, i.e., $\Theta = \{w_0, w_1, ..., w_n, v_{1,1}, ..., v_{n,k}\} = \{w_0, \mathbf{w}, \boldsymbol{V}\}$.

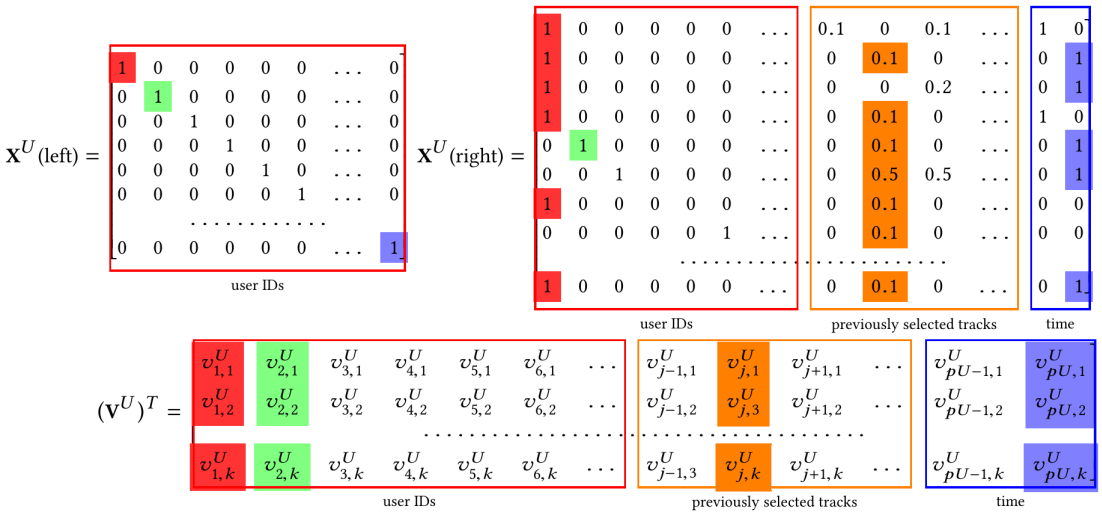


Figure 6.2: The structure of sparse input matrix and dense embedding matrix for user-fields. $\mathbf{X}^U$ (left) denotes the user-field matrix with only user IDs; $\mathbf{X}^U$ (right) denotes the user-field matrix with both user IDs and additional context variables; $(\mathbf{V}^U)^T$ is the transpose of $\mathbf{V}^U$.

The computation of Eq.(6.1) is $O(k \cdot (pU + pI))$. Note that (1) Eq.(6.1) only models interactions between user- and item-fields, which is different from factorization machines (FM) (Rendle, 2012) modeling all pairwise interactions; (2) we empirically find that f_{BGD} with Eq.(6.1) shows slightly better performance than f_{BGD} with FM. That means, the pairwise interaction of FM may bring additional noise in practice when there are many useless features. Hence, in the following, we introduce f_{BGD} based on Eq.(6.1).

6.3.2 Optimization with BGD

The task of item recommendation is to estimate parameters of the above scoring function, which is then used for item ranking. Thus, we propose to optimize a popular squared loss (i.e., Eq. (2.18)), which contains losses from all positive and negative examples.

$$J(\Theta) = \underbrace{\sum_{(u,i) \in S} \alpha_{ui}^{+} (y_{ui}^{+} - \hat{y}_{ui})^2}_{J_P(\Theta)} + \underbrace{\sum_{(u,i) \in (U \times I) \setminus S} \alpha_{ui}^{-} (y_{ui}^{-} - \hat{y}_{ui})^2}_{J_M(\Theta)} \tag{6.2}$$

where $J_P(\Theta)$ and $J_M(\Theta)$ denotes the errors of positive and unobserved data, α_{ui}^{+} and α_{ui}^{-} are the weight functions. Unlike the SGD-based models typically fed

with the same order of magnitude positive and negative examples, the weighting scheme for BGD is important to solve the imbalanced-class problem.

Eq.(6.2) can be minimized by BGD, which computes the gradient of the objective function w.r.t. $\theta \in \Theta$ for the entire samples:

$$\theta \leftarrow \theta - \gamma \nabla_\theta J(\theta) \tag{6.3}$$

where γ is the learning rate, and $\nabla_\theta J(\theta)$ is the first derivative (gradient) w.r.t. θ, given below:

$$\nabla_\theta J(\theta) = 2\Big(\sum_{(u,i)\in S} \alpha^+_{ui}(y^+_{ui} - \hat{y}_{ui}(\theta)) \nabla_\theta \hat{y}_{ui}(\theta) + \sum_{(u,i)\in (U\times I)\setminus S} \alpha^-_{ui}(y^-_{ui} - \hat{y}_{ui}(\theta)) \nabla_\theta \hat{y}_{ui}(\theta) \Big) \tag{6.4}$$

where $\nabla_\theta \hat{y}_{ui}(\theta)$ is the first derivative of $\hat{y}_{ui}(\theta)$.

6.3.3 Efficiency Challenge

As can be seen, the second term $J_M(\Theta)$ in Eq.(6.2) dominates the computational complexity. Specifically, the loss in Eq.(6.2) has almost $O(|U| \cdot |I| \cdot T_{\text{pred}})$ time because $|I^+_u| \ll |I|$, where I^+_u is the set of positive items for u. Similarly, updating a parameter, e.g., $v^U_{j,f}$, in Eq.(6.4) has almost $\tilde{n} \cdot |I| \cdot O(T_{\text{pred}})$ time, where $\tilde{n}$ is the number of user-fields that have a non-zero value of $x^U_{u,j}$. Accordingly, the total cost by iterating over all $v^U_{j,f}$ in $\mathbf{V}^U$ in each iteration becomes $pU \cdot n \cdot |I| \cdot O(T_{\text{pred}})$, where n is the average of all $\tilde{n}$. Clearly, the standard way to calculate the loss and gradient of BGD is computationally infeasible, because both $|U| \cdot |I|$ and $|pU| \cdot |I|$ can easily reach billion to trillion level (see Table 6.1). To handle the huge computation of BGD, most previous literature resorts to SGD with a negative item sampling technique for $(u, i) \in (U \times I) \setminus S$, such as in Pan et al. (2008); Rendle et al. (2009b); Zhang et al. (2013); Mikolov et al. (2013); Hong et al. (2013); Yuan et al. (2016b, 2017); Wang et al. (2017a), where the efficiency and effectiveness of learning algorithms depend highly on the size and quality of sampled data. We defer the discussion of these sampling methods to the experiments section.

6.4 f_{BGD}

The main contribution in this section is to derive an efficient BGD for general recommendation problems.

6.4.1 Partition of the BGD Loss

As analyzed above, the major computation lies in the minimization of the second term in Eq.(6.2). In implicit feedback scenarios, each user-field u has its standalone unobserved item-field set I_u^-. As such, an optimization algorithm basically needs to iterate through all elements in I_u^-, and repeat the operation for all $u \in U$, which produces in the main computational bottleneck. To solve the problem, we first conduct a partition operation on the standard loss, which serves as a prerequisite for the efficient computation. The key idea[1] is that the prediction errors of unobserved data can be naturally reformulated as the *residual* between the errors on all (u, i) pairs and that of observed (u, i) pairs.

$$J_M(\Theta) = \sum_{u \in U} \sum_{i \in I} \alpha_{ui}^- (y_{ui}^- - \hat{y}_{ui}(\theta))^2 - \sum_{(u,i) \in S} \alpha_{ui}^- (y_{ui}^- - \hat{y}_{ui}(\theta))^2 \tag{6.5}$$

By plugging Eq.(6.5) into Eq.(6.2), we obtain a new objective function that has a more clear structure — the correspondence relation between u and I_u^- is eliminated. To further simplify it, we merge the two terms that contain observed (u, i) pairs into a single term. Note that y_{ui}^+, y_{ui}^-, α_{ui}^+, α_i^- are independent of $\theta \in \Theta$.

$$\begin{aligned} J(\Theta) = & \underbrace{- \sum_{(u,i) \in S} \frac{\alpha_{ui}^+ \alpha_{ui}^- (y_{ui}^+ - y_{ui}^-)^2}{(\alpha_{ui}^+ - \alpha_{ui}^-)^2}}_{\text{const}} + \underbrace{\sum_{u \in U} \sum_{i \in I} \alpha_{ui}^- (y_{ui}^- - \hat{y}_{ui}(\theta))^2}_{J_A(\Theta)} \\ & + \underbrace{\sum_{(u,i) \in S} (\alpha_{ui}^+ - \alpha_{ui}^-) \big(\hat{y}_{ui}(\theta) - \frac{\alpha_{ui}^+ r_{ui}^+ - \alpha_{ui}^- r_{ui}^-}{\alpha_{ui}^+ - \alpha_{ui}^-} \big)^2}_{J_P(\Theta)} \end{aligned} \tag{6.6}$$

where $J_A(\Theta)$ and $J_P(\Theta)$ denote the loss for all (u, i) pairs and positive (u, i) pairs respectively; const denotes a Θ-invariant constant value. It can be seen that the loss of unobserved data has been eliminated. Instead, the new computation complexity lies in $\widetilde{J}_A(\Theta)$, which is part of $J_A(\Theta)$, defined as:

$$\widetilde{J}_A(\Theta) = \sum_{u \in U} \sum_{i \in I} \alpha_{ui}^- \hat{y}_{ui}(\theta)^2 - 2r^- \sum_{u \in U} \sum_{i \in I} \alpha_{ui}^- \hat{y}_{ui}(\theta) \tag{6.7}$$

Now we proceed to introduce the second key design that is based on the commutative property of nested sums and the constant-pullout operation. So far, we have ignored how $\hat{y}_{ui}(\theta)$ is computed. In the following, first we assume that

[1] The idea is similar to that used in He et al. (2016c), which can only solve the basic matrix factorization model based on a different optimization method (i.e., alternative least square).

$\hat{y}_{ui}(\theta)$ can be denoted by a dot product between a compressed user- and item-field embedding vector. Then we show how to construct the form of dot product for Eq. (6.1) and other embedding models.

Assuming $\hat{y}_{ui}(\theta) = \langle \mathbf{p}_u, \mathbf{q}_i \rangle = \sum_{f=1}^{g} p_{u,f} q_{i,f}$, Eq. (6.7) can be rewritten as:

$$\widetilde{J}_A(\Theta) = \sum_{u \in U} \sum_{i \in I} \alpha_{ui}^{-} \sum_{f=1}^{g} p_{u,f} q_{i,f} \sum_{f'=1}^{g} p_{u,f'} q_{i,f'} - 2r^{-} \sum_{u \in U} \sum_{i \in I} \alpha_{ui}^{-} \sum_{f=1}^{g} p_{u,f} q_{i,f} \quad (6.8)$$

where we observe that there exists a very nice structure in above equation — if α_{ui}^{-} is a constant value or a value only associates with u or i but not (u, i) pair. Considering that there is no observed (u, i) interaction in unlabeled examples, it is reasonable to set α_{ui}^{-} as α_{u}^{-} or α_{i}^{-}. The simplified weight design is a necessary condition for efficient optimization in the following. Here we continue to discuss the algorithm, assuming $\alpha_{ui}^{-} = \alpha_{i}^{-}$, and later show how to design a good weighting scheme. We can now show the rearrangement manipulation for the summation operator and "constant" variables in the dot product as below:

$$\widetilde{J}_A(\Theta) = \sum_{f=1}^{g} \sum_{f'=1}^{g} \sum_{u \in U} p_{u,f} p_{u,f'} \sum_{i \in I} \alpha_{i}^{-} q_{i,f} q_{i,f'} - 2r^{-} \sum_{f=1}^{g} \sum_{u \in U} p_{u,f} \sum_{i \in I} \alpha_{i}^{-} q_{i,f} \quad (6.9)$$

The rearrangement of nested sums in Eq.(6.9) is the key transformation that allows the fast optimization of f_{BGD}, as the major computation — the $\sum_{i \in I} \alpha_{i}^{-} q_{i,f} q_{i,f'}$ and $\sum_{i \in I} \alpha_{i}^{-} q_{i,f}$ terms that iterate over all $i \in I$ — are independent of u. In other words, we can achieve a significant speed-up (i.e. from $O(|U||I|g)$ in Eq.(6.8) to $O((|U| + |I|)g^2)$ in Eq.(6.9)) by pre-computing them. Optimization details regarding the gradient computation are given in Section 6.4.3.

6.4.2 Constructing a Dot Product Structure

As discussed above, we need to construct the dot product function for embedding models if they are not an explicit dot product structure. To transform Eq. (6.1) to the dot product structure $\langle \mathbf{p}_u, \mathbf{q}_i \rangle$, we just need to decompose it as[1]:

$$\sum_{f=1}^{g} p_{u,f} q_{i,f} \quad (6.10)$$

[1]The decomposition idea here is inspired by Rendle and Freudenthaler (2014), which solves the sampling problem.

where $g = k + 2$ and

$$\begin{aligned} p_{u,f} &= \sum_{j=1}^{pU} x^U_{u,j} v^U_{j,f} \ , & q_{i,f} &= \sum_{j=1}^{pI} x^I_{i,j} v^I_{j,f} \\ p_{u,k+1} &= w_0 + \sum_{j=1}^{pU} w_j x^U_{u,j} \ , & q_{i,k+1} &= 1 \\ p_{u,k+2} &= 1 \ , & q_{i,k+2} &= \sum_{j=pU+1}^{pU+pI} w_j x^I_{i,j-pU} \end{aligned} \tag{6.11}$$

As we can see, Eq. (6.11) is mathematically equal to Eq. (6.1). In fact, we notice that the dot product structure widely exists in many embedding modes. Here, we show the dot product structure in another two popular models:

(1) Regarding the basic matrix factorization (MF) (Koren et al., 2009; Mikolov et al., 2013) mode (also see Eq. (2.2) in Chapter 2 for more details), it is defined as follows in the context of this chapter:

$$\hat{y}_{ui}(\theta) = \sum_{f=1}^{k} v^U_{j,f} v^I_{j',f} \tag{6.12}$$

clearly, MF is an an explicit dot product structure. If we express it by $\sum_{f=1}^{g} p_{u,f} q_{i,f}$, we have $g = k$ and

$$p_{u,f} = v^U_{j,f} \ , \quad q_{i,f} = v^I_{j,f} \tag{6.13}$$

(2)The Pairwise Interaction Tensor Factorization (PITF) (Rendle and Schmidt-Thieme, 2010) is defined as follows in the context of this chapter:

$$\hat{y}_{ui}(\theta) = \sum_{f=1}^{k} v^U_{j,f} v^V_{j',f} v + \sum_{f=1}^{k} v^U_{j,f} v^I_{j'',f} + \sum_{f=1}^{k} v^V_{j',f} v^I_{j'',f} \tag{6.14}$$

If we express it by $\sum_{f=1}^{g} p_{uv,f} q_{i,f}$, we have $g = k + 1$ and

$$\begin{aligned} p_{uv,f} &= v^U_{j,f} + v^V_{j',f} \ , & q_{i,f} &= v^I_{j'',f} \\ p_{uv,fk1} &= \sum_{f=1}^{k} v^U_{j,f} v^V_{j',f} \ , & q_{i,k+1} &= 1 \end{aligned} \tag{6.15}$$

6.4.3 Efficient Gradient

Once we have achieved the efficient loss function, it is trivial to calculate the gradient computation. Here, we show the gradient computation of Eq. (6.9). By a similar transformation, f_{BGD} can be directly extended to MF and PITF. We

define the $S^q_{ff'}$ and S^q_f caches as $S^q_{ff'} = \sum_{i\in I} \alpha_i^- q_{i,f} q_{i,f'}$ and $S^q_f = \sum_{i\in I} \alpha_i^- q_{i,f}$ respectively, which can be pre-computed and used for the update of all user-field related parameters, i.e. $w_j (j < pU)$ and $v^U_{j,f}$. We now present the formulation for the gradient calculation.

$$\nabla_\theta \widetilde{J}_A(\theta) = 2\sum_{f=1}^{g}\sum_{f'=1}^{g} S^q_{ff'} \sum_{u\in U} p_{u,f'} \nabla_\theta p_{u,f} - 2r^- \sum_{f=1}^{g} S^q_f \sum_{u\in U} \nabla_\theta p_{u,f} \tag{6.16}$$

where

$$\nabla_{w_j} p_{u,f} = \begin{cases} x^U_{u,j} & f = k+1 \\ 0 & \text{otherwise} \end{cases}, \quad \nabla_{v^U_{j,f}} p_{u,f} = \begin{cases} x^U_{u,j} & f \le k \\ 0 & \text{otherwise} \end{cases} \tag{6.17}$$

First, it is clear that when computing both $\nabla_{v^U_{j,f}} \widetilde{J}_A(\theta)$ and $\nabla_{w_j} \widetilde{J}_A(\theta)$, one of nested sums in Eq.(6.16), (e.g., $\sum_{f=1}^{g}$) will vanish. Second, the computation of sums over user-fields $u \in U$ can be accelerated by only iterating over u where $x^U_{u,j} \neq 0$; Last, $p_{u,f'}$ is able to be precomputed to reduce the cost. Although $p_{u,f'}$ changes when updating θ^U, it can be updated in synchronization with the changes in θ^U, denoted by $\triangle\theta^U$.

$$p_{u,f'} \leftarrow p_{u,f'} + x^U_{u,j} \triangle\theta^U = p_{u,f'} - x^U_{u,j}\gamma\nabla_\theta J(\theta) 576s20032h \tag{6.18}$$

The total time complexity of $\nabla_\theta J_A(\theta)$ (or $\nabla_\theta \widetilde{J}_A(\theta)$) in one iteration for all parameters is $O(g^2(N(U) + N(I)))$, where $N(U)$ and $N(I)$ are the number of non-zero elements in $\mathbf{X}^U$ and $\mathbf{X}^I$. Finally, the efficient computation for θ^U is given as follows[1]:

$$\theta^U \leftarrow \theta^U - \gamma(\nabla_{\theta^U} J_A(\theta) + \nabla_{\theta^U} J_p(\theta)) \tag{6.19}$$

The optimization process of θ^I is near-symmetric to θ^U, except that the weighting scheme α_i^- is inside the item-field sum. For example, we define the $S^p_{ff'}$ and S^p_f caches as $S^p_{ff'} = \sum_{u\in U} p_{u,f} p_{u,f'}$ and $S^p_f = \sum_{u\in U} p_{u,f}$ respectively, and derive:

$$\nabla_\theta \widetilde{J}_A(\theta) = 2\sum_{f=1}^{g}\sum_{f'=1}^{g} S^p_{ff'} \sum_{i\in I} \alpha_i^- q_{i,f'} \nabla_\theta q_{i,f} - 2r^- \sum_{f=1}^{g} S^p_f \sum_{i\in I} \alpha_i^- \nabla_\theta q_{i,f} \tag{6.20}$$

where $\nabla_\theta q_{i,f}$ can be calculated by the same way. Algorithm 7 summarizes the accelerated algorithm for the efficient BGD.

[1] $\nabla_{\theta^U} J_p(\Theta)$ is the gradient of positive loss, which can be calculated by the standard way with $|I_u^+| \cdot O(T_{\text{pred}})$ complexity.

Algorithm 7 f_{BGD} Learning

1: **Input:** $\mathbf{X}^{\mathrm{S}}$, γ, λ_θ, Cache vectors $\mathbf{s}^q$, $\mathbf{s}^p$, Cache matrices $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{S}^q$, $\mathbf{S}^p$;
2: **Output:** $\Theta = (\mathbf{w}, \mathbf{V}^U, \mathbf{V}^I)$
3: Initialize Θ: $\mathbf{w} \leftarrow (0, ..., 0)$; $\mathbf{V}^U, \mathbf{V}^I \sim \mathcal{N}(0, 0.01)$;
4: **for** $e = 1, ...,$ maxiter **do**
5: **for** $f \in \{1, .., k+2\}$ **do**
6: **for** $u \in U$ **do**
7: Compute p_{ud}, store in $\mathbf{P}$ ($\mathbf{P} \in \mathbf{R}^{|U| \times (k+2)}$)
8: **end for**
9: Repeat line 6 to 8 for item-fields
10: **end for**
11: **for** $f \in \{1, .., k+2\}$ **do**
12: Compute S^q_f, store in $\mathbf{s}^q$, ($\mathbf{s}^q \in \mathbf{R}^{(k+2)}$)
13: **for** $f' \in \{1, .., k+2\}$ **do**
14: Compute $S^q_{ff'}$, store in $\mathbf{S}^q$, ($\mathbf{S}^q \in \mathbf{R}^{(k+2)\times(k+2)}$)
15: **end for**
16: **end for**
17: **for** $j \in \{1, .., pU\}$ **do**
18: **for** $f \in \{1, .., k+2\}$ **do**
19: Compute $\nabla_{\theta^U} J_A(\Theta)$,$\nabla_{\theta^U} J_p(\Theta)$
20: Update θ^U as in Eq.(6.19)
21: Update $p_{u,f'}$ as in Eq.(6.18)
22: **end for**
23: **end for**
24: Repeat line 11 to 23 for item-fields
25: **end for**

6.4.4 Effective Weighting on Missing Data

In the previous section, we provided the basic description of the speed-up process for BGD. In what follows, we take a closer look into the tailored item-wise weighting scheme on training data.

First, for the weight of a positive instance α^+_{ui}, any reasonable weighting scheme could be adopted in f_{BGD} which does not affect the analyzed computation. For example, it is convenient to design a scheme by accounting for users' observation frequency information such as in Hu et al. (2008). In this work, we simply set $\alpha^+_{ui} = 1$ for all observed entries, since each (u, i) pair has only one entry in our data set. However, in terms of α^-_{ui}, it is non-trivial to assign an individualized weight α^-_{ui} for each unobserved feedback, due to the high space complexity

of $O(|U| \cdot |I|)$ for storing them. Hence, most previous works (Hu et al., 2008; Devooght et al., 2015; Volkovs and Yu, 2015; Pilászy et al., 2010) tend to employ a simple uniform weight on all unobserved feedback

As has been discussed, either a user-wise ($\alpha^-_{ui} = \alpha^-_u$) or an item-wise ($\alpha^-_{ui} = \alpha^-_i$) weighting scheme can guarantee the efficient optimization of BGD. However, we did not observe significantly improved performance by using the user-wise weighting scheme. Hence, we focus on introducing the item-wise weighting scheme. Our implementation is originally motivated by the popularity-based negative sampling technique in the skip-gram model (Mikolov et al., 2013) for the natural language processing domain. The basic idea is that high-frequency items in unobserved feedback are more likely to be truly negative, as they are more likely to be exposed to users. While similar ideas have also been considered in previous works (Hidasi et al., 2015; Yuan et al., 2016b, 2017), these methods typically either employ an item frequency based oversampling scheme, which is tailored for the SGD/MGD or optimize a simple MF model only (Pilászy et al., 2010), which is a special case of our algorithm. Evidently, the sampling techniques do not naturally fit our model, as f_{BGD} takes for full batch training data into account instead of sampled data. Therefore, we believe that an item frequency based weighting scheme is a more reasonable approach for our optimization setting. Specifically, we assign a larger weight for the unobserved data with high item frequency, and a smaller weight otherwise.

$$\alpha^-_i = \alpha_0 \frac{(e^{\frac{p_i}{|S|}} - 1)^\rho}{\sum_{i=1}^{|I|}(e^{\frac{p_i}{|S|}} - 1)^\rho} \tag{6.21}$$

where p_i denotes the occurrence frequency for item i, given by the number of observations in the observed set S, and α_0 controls the overall weight of unobserved data to solve the imbalanced-class problem as mentioned in the introduction section. The exponent ρ controls the weight distribution, which should be tuned for different data sets. Note that by setting $\rho = 0$ the item-wise weighting scheme is reduced to a uniform one, as discussed before.

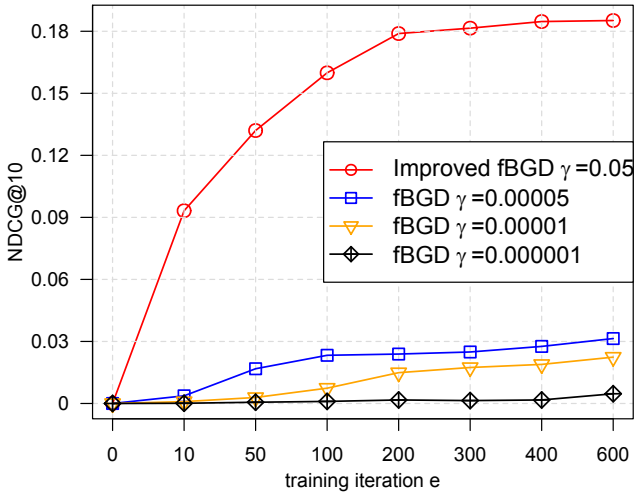


Figure 6.3: Performance of the improved f_{BGD} (Section 6.5.2) and standard f_{BGD} on Last.fm with four features. For the standard f_{BGD}, some gradients will be evaluated as infinite (NaN) when $\gamma > 5 \times 10^{-5}$. As can be seen, f_{BGD} with vanishing gradient performs poorly on Last.fm even by carefully tuning the learning rate.

6.5 Improved f_{BGD}

So far, we have discussed the f_{BGD} framework and found that by fine-tuning its hyper-parameters we can achieve a promising performance for the classical CF setting (with only user ID and item ID as features), in terms of both efficiency and effectiveness. However, we observe unreliable results on settings where there are rich contextual features besides user ID and item ID, as evidenced in Figure 6.3. A novel contribution here is to reveal why unstable gradient issues will occur for f_{BGD} when modelling more features.

6.5.1 Gradient Instability Issue in CF Settings

While the unstable gradient (i.e., the gradient exploding and vanishing) problem has been observed when training deep neural networks (Le and Zuidema, 2016), our predictive model is a shallow embedding (with one hidden layer only). Therefore, the cause of the unstable gradient issue in our case is fundamentally different from that in the existing deep layer models, in the sense that in deep models unstable gradients occur mainly due to cumulative multiplying of small/big numbers from previous layers, whereas in f_{BGD} it is caused by the large batched summa-

tion of sparse features. We expect the following theoretical analysis and solution could provide practical guidelines for the future development of recommender systems based on the BGD framework.

To understand the abnormal behavior of gradient instability, we need to first revisit the form of gradients. We continue to take the derivation of $v^U_{j,f}$ as an example.

$$\nabla_{v^U_{j,g}} J_A(\theta) = 2 \sum_{f'=-1}^{k} S^q_{ff'} \sum_{u \in U} p_{u,f'} x^U_{u,j} - 2r^- S^q_f \sum_{u \in U} x^U_{u,j} \tag{6.22}$$

$$\nabla_{v^U_{j,f}} J_P(\theta) = 2 \sum_{u \in U} \sum_{i \in I^+_u} (\alpha^+_{ui} - \alpha^-_i)\big(\hat{y}_{ui}(\theta) - \frac{\alpha^+_{ui} y^+_{ui} - \alpha^-_i y^-_{ui}}{\alpha^+_{ui} - \alpha^-_i}\big) q_{i,f} x^U_{u,j} \tag{6.23}$$

Due to the data sparsity in CF settings, to compute $\sum_{u \in U} x^U_{u,j}$ we only need to consider the user-field $u \in U$ that has a non-zero $x^U_{u,j}$ (note that for a feature j, most user-fields u have $x^U_{u,j}$ equal to zero which can be safely ignored). Let l_j be the number of non-zero elements in the j-th column of $\mathbf{X}^U$. In Figure 6.2, we highlight the non-zero elements in $\mathbf{X}^U$ as red, green and blue for $j = \{1, 2, pU\}$ respectively. As can be seen, the number of non-zero elements l_j in $\mathbf{X}^U$(left), defined as $l_j(\mathbf{X}^U(\text{left}))$, equals 1 for all $j \leq pU$. $l_j(\mathbf{X}^U(\text{right}))$, however, is associated with j as $\mathbf{X}^U$(right) stores multiple sparse features. For example, in Figure 6.2 we have

$$l_j(\mathbf{X}^U(\text{right})) = \begin{cases} 6, & j = 1 \\ 1, & j = 2 \\ 5, & j = pU \end{cases} \tag{6.24}$$

In real-world data sets, the number of rows in $\mathbf{X}^U$ (i.e. $|U|$) can easily scale to that of million or even billion rows and, therefore, it is very likely that l_j has distinct magnitudes for a different column j. Moreover, we observe that in Eq.(6.23) there is another summation $\sum_{i \in I^+_u}$, which represents the size of observed feedback for u. The component value of $\sum_{u \in U} \sum_{i \in I^+_u} x^U_{u,j}$ in Eq.(6.23) varies from 1 to $|U| \cdot |I|$, assuming $\mathbf{X}^U$ is a binary matrix. This indicates the value of Eq.(6.23) may be unstable: $\nabla_{v^U_{j,f}} J_P(\theta)$ can be too large for a denser feature j that is accompanied by a large $\sum_{u \in U} \sum_{i \in I^+_u} x^U_{u,j}$ (e.g., $= 10^6$), while it may be too small for a sparser feature with a small $\sum_{u \in U} \sum_{i \in I^+_u} x^U_{u,j}$(e.g., $= 1$). Similarly, the overall gradient $\nabla_\theta J(\theta)$ in Eq.(6.19) has the same unstable problem. In this

case, a uniform learning rate γ is no longer suitable because $\nabla_\theta J(\theta)$ with a larger $\sum_{u \in U} \sum_{i \in I_u^+} x_{u,j}^U$ is likely to explode (i.e. $\nabla_\theta J(\theta) = \text{NaN}$) if using a large γ, while $\nabla_\theta J(\theta)$ with a smaller $\sum_{u \in U} \sum_{i \in I_u^+} x_{u,j}^U$ may vanish (i.e. $\nabla_\theta J(\theta) \approx 0$) if using a small γ. Generally, it is hard or even impossible to find a medium learning rate that balances reasonably well in both conditions. To gain more insight into the performance of f_{BGD} with unstable gradients, we show results with different learning rates in Figure 6.3.

Interestingly, we empirically find that on data sets with only user ID and item ID, the gradient instability problem may be avoided by carefully tuning γ. In other words, by many trials with different learning rates, f_{BGD} is able to offer reasonable results. However, on data sets with more feature variables (e.g., Last.fm), the outputs of f_{BGD} are prone to the NaN error. This is because in the pure CF setting, the nested summation $\sum_{u \in U}$ can be dropped since only a user ID variable is available. As such, although the gradient instability issue may still happen because of $\sum_{i \in I_u^+}$, it is less severe because the value of $|I_u^+|$ is much smaller than that of $l_j \cdot |I_u^+|$. The same analysis is also applicable to the parameter estimation associated with item-fields.

6.5.2 Solving the Unstable Gradient Problem

The above theoretical analysis for gradient estimation over all data suggests that the same learning rate does not hold for all model parameters due to the large batched summation in sparse data. Analytically, by assigning a specific learning rate for each parameter update, we can contain the unstable gradient to a certain extent. Considering this, one possible solution like the one proposed in Adagrad (Duchi et al., 2011), would be to apply a heavy penalisation by using a smaller learning rate for $\nabla_\theta J(\theta)$ if θ has larger historical gradients during the past training iterations, and vice versa.

Considering that Adagrad is mostly applied to stochastic gradient method for accelerating the convergence of deep neural models, here we only demonstrate how it can be applied on the full gradient method to address the gradient instability issue. Denoting γ_t as the learning rate for the t-th update, we then assign a

personalized learning rate for each parameter θ:

$$\gamma_t(\theta) = \frac{\gamma}{G_t(\theta)}, \quad G_t(\theta) = \begin{cases} \nabla_\theta J(\theta)_t + \epsilon & G_t(\theta) = 0 \\ \sqrt{\sum_{t=1}(\nabla_\theta J(\theta)_t)^2} & G_t(\theta) \neq 0 \end{cases} \tag{6.25}$$

where $\nabla_\theta J(\theta)_t$ is the gradient w.r.t. θ for the $t-$th update, $G_T(\theta)$ is the accumulation of historical gradients, and ϵ is a smoothing term to avoid division by zero, set as 10^{-4}. The overall algorithm of improved f_{BGD} can be implemented by replacing γ in Eq.(6.19) and Eq.(6.18) with the new $\gamma_t(\theta)$. In our experiments, we found that the adaptive learning rate strategy not only addresses the gradient instability issue for more feature settings, but also works well in the basic CF setting. However, to provide a fair comparison, we also evaluated in Section 6.6 baselines with both fixed and adaptive learning rates, but found that the adaptive learning scheme used for f_{BGD} had little effect on the end result compared with a fixed one. This suggests that the same unstable gradient issue is not such a frequently encountered issue in SGD-based shallow embedding models. It may also explain why in these original work (Chen et al., 2012; Rendle, 2012; Qiang et al., 2013; Yuan et al., 2016b, 2017; Wang et al., 2017a) a regular (non-adaptive) learning rate was typically adopted for optimization. However, providing further analysis on this matter falls beyond the scope of this work.

6.6 Experiments

As the key contribution of this work is in the deriving of an efficient batch gradient algorithm that learns from whole implicit data instead of sampled data, we conduct experiments to study its prediction accuracy in contrast with negative sampling based SGD baselines, and examine its efficiency in contrast with original BGD.

6.6.1 Experimental Settings

6.6.1.1 Datasets

For our evaluation, we use two data sets, namely, Yahoo Music[1] and Last.fm[2]. First, we perform a the basic pre-processing on the two data sets: to speed up the

[1] http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=2
[2] http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/

experiments we follow the same procedure as in Christakopoulou and Banerjee (2015); Yuan et al. (2016b) by randomly drawing a subset of users from the user-pool of the Yahoo Music data set; on the Last.fm data set, similarly as in Weston et al. (2012) we extract the latest one-week actions per user via the timestamp, and consider two tracks played by the same user as "consecutive" if they are played within 90 minutes. In our experiments Yahoo music is used as the traditional CF data set, where user- and item-field contain only user ID and item ID. Last.fm is used as a context-aware (or next-item) recommendation data set, where each user-field contains a user ID and her previously played music track and each item-field contains a song ID and the artist information. Furthermore, on Last.fm similarly to Rendle et al. (2009b), we remove those cases of users that have listened to fewer than ten tracks, and item-fields that have been selected less than 5 user-fields. This is because the high sparsity in the original data set makes it difficult to evaluate recommendation algorithms (e.g., some user-fields only have one action). Note that, cold users and items on Yahoo have been pruned by the official provider. Table 6.1 provides basic statistics on the data set, after the preprocessing step.

6.6.1.2 Baselines and Evaluation Protocols

In this section, we compare f_{BGD} with a variety of state-of-the-art embedding-based models (i.e., Most Popular (MP) (see Chapter 3), SGD, Pairwise Ranking FM (PRFM) (Yuan et al., 2016a; Qiang et al., 2013), lambdaFM, BoostFM) that are specifically designed for prediction from implicit data. MP and PRFM are omitted since they have been described in previous chapters.

- **SGD** (Chen et al., 2012; Rendle, 2012): This baseline uses SGD to optimize the point-wise regression loss function with a feature-based factorization model, such as FM or Eq.(6.1). We report the results of Eq.(6.1) since it performs slightly better than FM on Last.fm[1]. The "negative" items used for training are uniformly sampled from the unobserved item set. To show the impact of negative sampling, we vary the size of negative actions for

[1]The better accuracy is probably because Eq.(6.1) only models useful feature interactions between user- and item-field, ignoring them among the same field which are typically redundant Juan et al. (2016).

each positive one. E.g., SGD×16 denotes the positive-to-negative ratio is 1:16, which is consistent with the results shown in Figure 6.1.

- **LambdaFM** (Yuan et al., 2016b): LambdaFM has been reported as a state-of-the-art item recommendation baseline in Wang et al. (2017a). The authors proposed three Lambda based sampling methods for optimizing ranking metrics, such as NDCG. We only investigate LambdaFM with the item popularity-based static sampler since the training time is much shorter than the other two dynamic samplers.

- **BoostFM** (Yuan et al., 2017): A recently published implicit recommendation model inspired by boosting framework. The authors proposed two models, namely B.WPFM and B.WLFM, by adopting different samplers for the component recommenders. We present results using both models. Please note that B.WPFM is able to mimic AdaBPR (Liu et al.) on Yahoo with only user and item variables. B.WLFM is equivalent to LambdaFM when there is only one component recommender.

We adopt the leave-one-out evaluation protocol as in Rendle et al. (2009b); He et al. (2017). On Last.fm, the latest action per user-field u is held out for prediction, and all models are trained on the remaining data. While for the Yahoo Music data set, we select randomly one action per u for prediction, since there is no timestamp information. This results in a disjoint training set $S_{\text{training}} = S \backslash S_{\text{test}}$ and test set S_{test}. To assess the recommendation accuracy of f_{BGD}, we adopt two standard ranking metrics: Normalized Discounted Cumulative Gain (NDCG) and Mean Reciprocal Rank (MRR) respectively. We truncate the ranked list at 10 and 50 for both metrics, denoted by NDCG@10 and NDCG@50, MRR@10 and MRR@50. For each metric, we first calculate the accuracy per u from S_{test}, and then obtain the average performance over all $u \in U$.

6.6.1.3 Experimental Reproducibility

For a fair comparison purpose, all reported results use an embedding dimension of $k = 20$ similarly as in Yuan et al. (2016b) if not explicitly declared. Results for $k = 10, 50, 100, 200$ show similar behavior but are omitted for saving. Regarding

Table 6.1: Basic statistics of the datasets. The density on Yahoo and Last.fm is 0.28% and 0.033% respectively.

Datasets	$\|U\|$	$\|I\|$	pU	pI	$\|S\|$
Yahoo	200,000	136,736	200,000	136,736	76,344,627
Last.fm	62,594	57,975	64,691	75,042	1,271,769

Table 6.2: Comparison of implicit embedding models.

Models	Sampler	Ratio	Optimizer	Loss
SGD×1	Uniform	1:1	SGD	LS
SGD×256	Uniform	1:256	SGD	LS
PRFM	Uniform	1:1	SGD	LtR
LambdaFM	Static	1:1	SGD	LtR
B.WPFM	Uniform	1:1	SGD	LtR
B.WLFM	Static	1:1	SGD	LtR
f_{BGD}	-	-	BGD	LS

"Uniform","Static" and "Dynamic" are short for a uniform, static and dynamic sampler respectively. Static sampler means the sampling distribution of negative items is defined before training and keeps unchanged during the whole optimization process. Dynamic sampler changes the sampling distribution of negative items according to the current state of learning. "Ratio" represents the positive-to-negative item ratio. "LS", "LOG", "LtR" and "MiniMax" are short for least square, logistic, learning-to-rank (LtR) and minimax loss function respectively.

f_{BGD}, the other hyperparameters on both datasets are: the learning rate $\gamma = 0.05$, regularization $\lambda_\theta = 0.01$, maxiter $= [200, 400]$. Empirically, we find that f_{BGD} will nearly converge in around 200 iterations. After that, more iterations will only bring very slight improvements (see Figure 6.3). The weighting hyperparameters α and ρ are tuned according to specific data, which is shown later. Regarding other baselines, we tune γ and λ_θ from $0.001, 0.005, 0.01, 0.05, 0.1$ to find the optimal performance. For BoostFm, the number of component recommenders is set to the default value, i.e. 10. ρ for LambdaFM and B.WLFM is tuned from $0.1, 0.3, 0.5, 0.8, 1.0$ according to Yuan et al. (2016b, 2017).

6.6.2 Performance Evaluation

6.6.2.1 Model Comparison

In what follows, we provide the overall performance of all models on the two data sets in Table 6.3 and 6.4. We also summarize the characteristics of the different models in Table 6.2. Our initial observation is that f_{BGD} outperforms all other

Table 6.3: Accuracy evaluation on Yahoo. f^{-}_{BGD} denotes f_{BGD} with a uniform weight ($\rho = 0$). The training iterations of f^{-}_{BGD} is 400. For each measure, the best results for SGD and all models are indicated in bold, which also applies to Table 6.4.

Methods	NDCG@10	NDCG@50	MRR@10	MRR@50
MP	0.0046	0.0099	0.0031	0.0042
PRFM	0.0178	0.0346	0.0124	0.0157
LambdaFM	0.0200	0.0380	0.0140	0.0176
B.WPFM	0.0186	0.0358	0.0129	0.0163
B.WLFM	0.0216	0.0397	0.0152	0.0188
SGD×1	0.007	0.0172	0.0045	0.0065
SGD×4	0.0133	0.0292	0.009	0.0122
SGD×16	0.0186	0.0355	0.0132	0.0166
SGD×64	**0.0197**	**0.0364**	**0.0139**	**0.0172**
SGD×256	0.0193	0.0353	0.0139	0.0169
f^{-}_{BGD}	0.0223	0.0395	0.0160	0.0194
f_{BGD}	**0.0237**	**0.0406**	**0.0166**	**0.0200**

methods in terms of prediction quality. We attribute this finding on two aspects: (1) the optimization of each model parameter in f_{BGD} makes use of the whole implicit data, whereas the other SGD-based models only use a fraction of sampled data. In other words, the convergence of SGD is suboptimal relative to the whole training data; (2) the proposed weighting strategy can address effectively the imbalanced-class problem in implicit data, as well as assign item-wise penalties for further prediction improvement. The proposed weighting strategy is an important component for f_{BGD} to obtain state-of-the-art performance. Next, we focus on investigating the first aspect that is related to our main contribution, and before concluding we address the following research question (RQ): **how can negative sampling strategies influence the performance of these SGD-based models in implicit data, regarding both prediction accuracy and efficiency?**

To show the impact of sampling size, we start from SGD×1 by increasing the number of sampled negative items corresponding to each positive one[1], denoted by SGD×n. Table 6.3 shows that SGD×64 > SGD×16 > SGD×4 > SGD×1, which indicates that the prediction accuracy of SGD is sensitive to the sampling size of

[1]Note that due to the loss function limitations, it generally does not hold by adding more than one sampled negative item for PRFM, LambdaFM, BoostFM and IRGAN.

Table 6.4: Accuracy evaluation on Last.fm.

Methods	NDCG@10	NDCG@50	MRR@10	MRR@50
MP	0.0014	0.0030	0.0010	0.0013
PRFM	0.1056	0.1738	0.0740	0.0883
LambdaFM	0.1312	0.1989	0.0950	0.1094
B.WPFM	0.1121	0.1827	0.0790	0.0938
B.WLFM	0.1462	0.2132	0.1069	0.1211
SGD×1	0.0436	0.0955	0.0285	0.0389
SGD×2	0.0511	0.1033	0.0341	0.0447
SGD×4	0.0565	**0.1055**	0.0389	0.0489
SGD×8	**0.0596**	0.1033	**0.0433**	**0.0520**
SGD×16	0.0535	0.0905	0.0390	0.0464
SGD×64	0.0360	0.0574	0.0263	0.0306
f^{-}_{BGD}	0.0801	0.1477	0.0541	0.0681
f_{BGD}	**0.1847**	**0.2477**	**0.1411**	**0.1548**

negative items. Specifically, one negative sample per positive item is insufficient to achieve optimal performance; sampling more negative items is beneficial but too many negative items may also hurt the performance, which can be seen from the result of SGD×256. Hence, the optimal sampling ratio has to be searched by performing many experimental trials, which is time-consuming in practice. Besides, although SGD×64 $>$[1] SGD×1 in prediction accuracy, the theoretical computation complexity of SGD×64 is about 32 times higher than SGD×1, which can be verified in Figure 6.1 (b). Similar observations can be made in Table 6.4 on the Last.fm data.

To show the impact of sampling distribution, we conduct a performance comparison of PRFM to LambdaFM and B.WPFM to B.WLFM. PRFM and LambdaFM share the same loss function, sampling ratio and SGD optimizer (Table 6.2) but give different prediction quality. The improved performance of LambdaFM over PRFM is due to the popularity-based static sampler, which in LambdaFM is more effective than the uniform one in PRFM, in terms of identifying 'good' negative examples (Yuan et al., 2016b). Similar observation and analysis apply to B.WPFM and B.WLFM.

With the above analysis, we have demonstrated the clear limitations of negative sampling in implicit data settings. The above analysis provide supports for

[1] In the remainder, '$>$' denotes 'outperform'.

Table 6.5: Accuracy evaluation on Last.fm by adding features. u, p, i and a denote user, last item (song), next item and artist respectively. Note u & p belong to the user-field, and i & a belong to the item-field. All hyper-parameters of f_{BGD} are fixed.

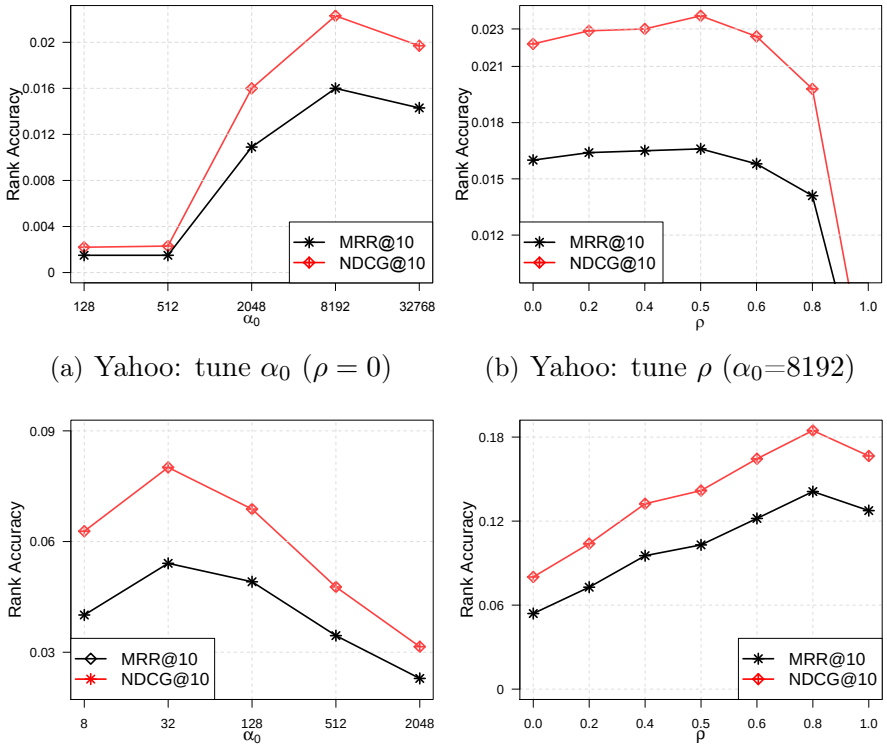
Features	NCDG@10	NCDG@50	MRR@10	MRR@50
(u, i)	0.0416	0.0837	0.0281	0.0365
(u, p, i)	0.1761	0.2378	0.1333	0.1467
(u, p, i, a)	0.1847	0.2477	0.1411	0.1548

our thesis **Statement (3)**. We now proceed to investigating the merits of f_{BGD} that does not rely on any kind of sampling strategies.

For a fair comparison, we employ a uniform weight for all negative items by setting ρ to 0, denoted by f^{-}_{BGD} . Thus, both f^{-}_{BGD} and SGD×n use the least square loss function but they differ in their optimizers: f^{-}_{BGD} adopts the batch gradient optimizer while SGD×n adopts SGD. Table 6.3 displays evidence that f^{-}_{BGD} outperforms SGD×64, which holds the best prediction among all SGD×n variants. Similar observations can be drawn from Table 6.4, where we note that f^{-}_{BGD} SGD×8. We argue that the improved performance of f^{-}_{BGD} over SGD is attributed to f^{-}_{BGD} ability to converge on the whole implicit data instead of a fraction of sampled data. That is, the optimization direction of f^{-}_{BGD} is likely to find a global or local minimum of the overall objective function (Eq.(6.2)), while the optimization direction of SGD is to find the optimum for each single instance (i.e. without $\sum$ in Eq.(6.2)). The optimal direction (i.e. reducing prediction errors) for each instance may not be the optimum for whole data. This is supported by the reported results, which indicate clearly the importance of optimizing model parameters by adopting the batch gradient method and with the whole implicit data. In what remains, we investigate the properties of f_{BGD} for the prediction in implicit data, regarding the weighting strategy, the capacity of modeling rich features, and its high efficiency.

6.6.2.2 Impact of f_{BGD} Weighting

Interestingly, we observe that f_{BGD} with a uniform weight (i.e. f^{-}_{BGD}) yields inferior results than PRFM in Table 6.4. The reason is because our f_{BGD} is built on the point-wise regression (quantitative) loss function, whereas implicit



(a) Yahoo: tune α_0 ($\rho = 0$)

(b) Yahoo: tune ρ (α_0=8192)

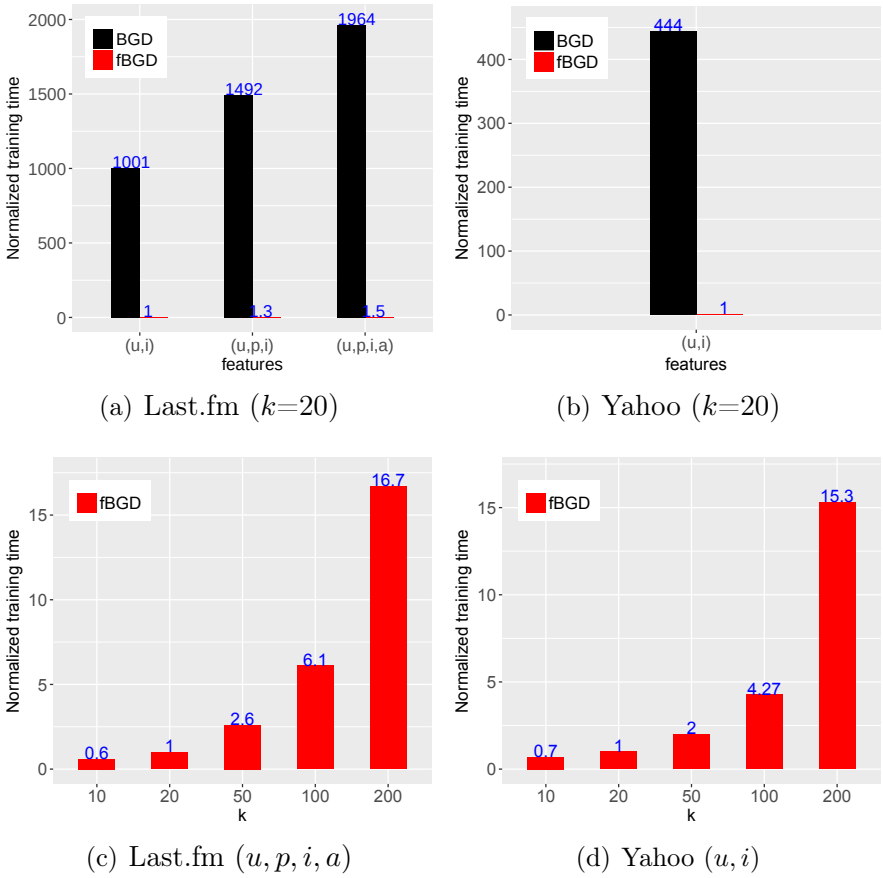
(c) Last.fm: tune α_0 ($\rho = 0$)

(d) Last.fm: tune ρ (α_0=32)

Figure 6.4: Impact of weighting parameters α_0 and ρ.

recommendation problem is actually regarded as an item ranking (qualitative) task. With a suitable ranking loss function, PRFM is able to optimize the ranking metrics more directly and thus is more effective. To pursue the state-of-the-art performance, we have implemented a simple yet effective weighting scheme in Section 6.4.4. Now we show how the weighting strategy impacts the performance of f_{BGD}.

Figure 6.4 shows the prediction quality by tuning α_0 and ρ in the weight function. As previously discussed, the overall coefficient α_0 largely impacts f_{BGD} as the amount of positive and "negative" feedback in the whole implicit data is highly unbalanced, the results of which are reflected in (a) and (c). We then adjust α_0 to the best value (in our case of $\rho = 0$), and vary ρ to investigate its impact on the performance, reflected in (b) and (d). It can be seen that a proper ρ improves the performance on both data sets. Moreover, when we compare the Yahoo data set to that of Last.fm, we notice a much larger improvement. The reason for



(a) Last.fm (k=20) (b) Yahoo (k=20)

(c) Last.fm (u, p, i, a) (d) Yahoo (u, i)

Figure 6.5: Training costs of f_{BGD} vs BGD. One unit in (a) and (b) represents 26 and 165 seconds respectively.

that is probably because the high-frequency (popular) music tracks for next-song recommendation (Last.fm) have a higher likelihood to be true negatives, whereas for the common item recommendation task (Yahoo) they are more likely to be a true positive. The results also provide insights as to why MP performs worse on Last.fm.

6.6.2.3 Impact of Adding Features

As previously discussed, we attempt to optimize a generic recommender model, which can generate recommendation in either traditional user-item based CF settings or context-aware settings. We experimentally compare and contrast our models, by gradually adding features for f_{BGD}, and report our findings in Table 6.5. As expected, we note that f_{BGD} performs significantly better with

(u, p, i) than (u, i) and that performance is further enhanced with (u, p, i, a) than (u, p, i). As a result, f_{BGD} achieves the best prediction accuracy with all features, demonstrating its power on feature engineering.

6.6.2.4 Efficiency

We show the relative training time of BGD and f_{BGD} per iteration on Figure 6.5 (a) and (b). First, (a) shows that with all features, the training time of BGD takes more than 10 hours in each iteration. This is clearly infeasible as BGD/f_{BGD} demands more than 100 iterations to converge on both datasets. In contrast, the training time of f_{BGD} is reduced to 26 seconds, which is more than 1000x faster than BGD. A similar observation is shown in (b). Further, considering that the theoretical complexity of f_{BGD} is subject to embedding dimension k, we also show the results regarding an increasing k on (c) and (d). The training time is almost linear with k rather than k^2, which potentially implies that the overall time complexity is dominated by the observed training data rather than the unobserved data.

6.7 Chapter Summary

In this work, we elaborated how to train a generic embedding model by a (full) batch gradient method and simplify the objective function and its gradients, whereby the large volume of computational cost due to massive unobserved feedback can be efficiently eliminated. We then pointed out the issues of gradient exploding and vanishing, which may be incurred due to the large batched summation of sparse features. A per-parameter learning scheme arises naturally to resolve the issue, once the key cause is seen. To seek state-of-the-art accuracy, we further implemented a simple yet effective item-wise weighting scheme for unknown items. Moreover, compared with many advanced baseline rankers, f_{BGD} is a regression model, through which the real-valued scores estimated will be more informative than those by ranking algorithms. This will make our method highly attractive for practical usage. The proposed batch gradient method can also be used in other predictive scenarios with positive-only data, e.g., training the word-embedding models (Xin et al., 2018), where word ID and context ID

can be simply replaced with user ID and item ID in this work without any change on the model.

This chapter includes the detailed investigation of the NS-based SGD methods and the whole data based BGD method. All points in our thesis statements have been empirically studied. Hence, this chapter can support all the four statements in this thesis.

Part IV

Deep Learning for Session-based Recommendation

In the last two parts, we mainly focus on shallow embedding models that consist of only one hidden layer. As is well known, the shallow models have limited expressive capacity. To model more complex preference relations, deep learning models has become the most important solutions in various domains of recommender systems. In this part, we investigate deep learning models in the field of session-based recommendation, which is also an implicit feedback scenario. Specifically, we introduce a novel sequential generative model that is based on an efficient convolutional network architecture. To show the recommendation quality of the proposed model, we study it on large-scale real-world recommendation datasets. Furthermore, we have also verified the efficiency advantage by comparing the proposed model with well-known baseline models.

Chapter 7

Deep Learning for Session-based recommendation

Session-based recommendation is an important task for real-world recommender systems. In recent two years, it has become increasingly popular due to the advancement of deep learning models. Compared with traditional Markov chain based models, deep learning models are more powerful in modeling complex sequential information (Tang and Wang, 2018).

Considering that recurrent neural networks (RNNs) based recommendation models have been extensively studied, we mainly investigate convolutional neural networks (CNNs) models for generating sequential recommendation. Recently, CNN sequential recommendation models have been introduced in this field. In this chapter, we first examine the typical sequential CNN recommender and show that both the generative model and network architecture are suboptimal when modeling long-range dependencies in the item sequence. To address the issues, we propose a new probabilistic generative model that is capable of learning high-level representation from both short- and long-range dependencies. We then introduce a network architecture that is formed as a stack of *holed* convolutional layers, where holes are used to increase the receptive fields to reduce parameters. Finally, we apply residual block structure to wrap each layer, with which the optimization for much deeper networks are feasible. The proposed generative model attains state-of-the-art accuracy with less training time in the next item recommendation task.

It is worth mentioning that due to the powerful expressive capacity of deep neural network (DNN), we model the implicit session data differently from previ-

ous work. That is, we do not directly compute the preference scores of items but generate the probability distribution of them and choose the next item with the highest probability value. The way of generating desired probability distribution is similar to van den Oord et al. (2016); Oord et al. (2016b,a); Kalchbrenner et al. (2016), which are used in very different research fields such as for voice, image generation and machine translation.

This chapter is mainly based on our recent work "A Simple but Hard-to-Beat Baseline for Session-based Recommendations" in http://cn.arxiv.org/abs/1808.05163.

7.1 Introduction

Leveraging sequences of item interactions (e.g., clicks or purchases) to improve real-world recommender systems has become increasingly popular in recent years. These sequences are automatically generated when users interact with online systems in sessions (e.g., shopping session, or music listening session). For example, users on Last.fm[1] or Soundcloud[2] typically enjoy a series of songs during a certain time period without any interruptions, i.e., a listening session. The set of songs played in one session usually have strong correlations (Cheng et al., 2017), e.g., sharing the same album, artist, or genre. Accordingly, a good recommender system is supposed to generate recommendations by taking advantage of these sequential patterns in the session.

A class of models often employed for these sequences of item interactions are the Recurrent Neural Networks (RNNs). RNNs typically generate a softmax output where high probabilities represent the most relevant recommendations. While effective, these RNN-based models, such as (Hidasi et al., 2015; Chatzis et al., 2017; Quadrana et al., 2017), depend on a hidden state of the entire past that prevents parallel computation within a sequence (Gehring et al., 2017). Thus their speed is limited in both training and evaluation.

By contrast, training CNNs does not depend on the computations of the previous time step and therefore allow parallelization over every element in a sequence. Inspired by the successful use of CNNs in image tasks, a newly proposed

[1] https://www.last.fm
[2] https://www.soundcloud.com

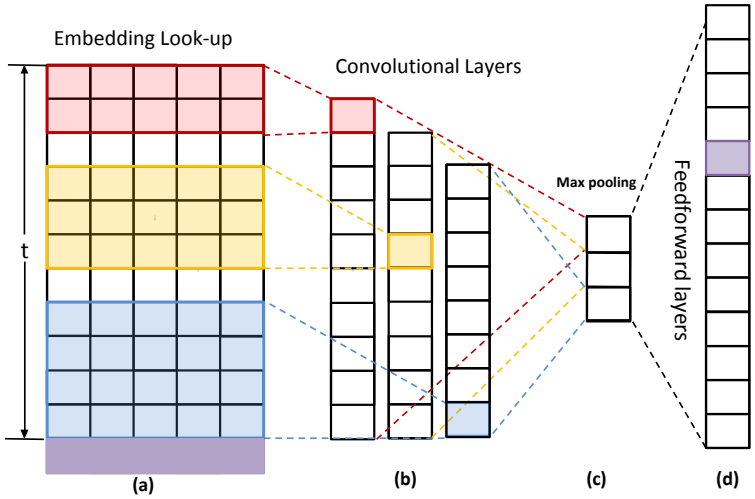


Figure 7.1: The core structure of Caser. The red, yellow and blue regions denotes a $2 \times k$, $3 \times k$ and $4 \times k$ convolution filter respectively, where $k = 5$. The purple row stands for the true next item.

sequential recommender, referred to as Caser (Tang and Wang, 2018), abandoned RNN structures, proposing instead a convolutional sequence embedding model, and demonstrated that the proposed CNN-based recommender gives comparable prediction accuracy with the RNN model in the top-N sequential recommendation task. The basic idea of the convolution processing is to treat the $t \times k$ embedding matrix as the "image" of the previous t items in k dimensional latent space and regard the sequential pattens as local features of the "image". A max pooling operation that only preserves the maximum value of the convolutional layer is performed to increase the receptive field, as well as dealing with the varying length of input sequences. Figure 7.1 depicts the key architecture of Caser.

Considering the training speed of networks, in this chapter we follow the path of sequential convolution techniques for the next item recommendation task. We show that the typical network architecture used in Caser have several obvious drawbacks — e.g.,: (1) the max pooling scheme that is safely used in computer vision may discard important position and recurrent signals when modeling long-range sequence data; (2) generating the softmax distribution only for the desired item fails to effectively use the compete set of dependencies. Both drawbacks become more severe as the length of the sessions and sequences increases. To address these issues, we introduce a fundamentally different CNN-based sequential

generative model that allows us to model the complex conditional distributions even in very long-range item sequences. To be more specific, first our generative model is designed to explicitly encode item inter-dependencies, which allows to directly estimate the distribution of the output sequence (rather than the desired item) over the raw item sequence. Second, instead of using inefficient huge filters, we stack the 1D dilated convolutional layers (Yu and Koltun, 2015) on top of each other to increase the receptive fields when modeling long-range dependencies. Thus no pooling layer is used in the proposed CNN structure. It is worth noting that although the dilated convolution was invented for dense prediction in image generation tasks (Chen et al., 2016; Yu and Koltun, 2015; Sercu and Goel, 2016), and has been applied in other fields (e.g., acoustic (Sercu and Goel, 2016; Oord et al., 2016a) and translation (Kalchbrenner et al., 2016; Strubell et al., 2017) tasks), it is yet unexplored in recommender systems with huge sparse data. Furthermore, to ease the optimization of the deep generative architecture, we propose using residual network to wrap convolutional layer(s) by residual block. To our knowledge, this is also the first work in terms of using residual learning for sequential recommendation. The combination of these choices enables us to tackle large-scale problems and attain state-of-the-art results in both short- and long-range sequential recommendation data sets. In summary, our main contributions in this chapter include a novel probabilistic generative model (Section 7.3.1) and a fundamentally different convolutional network architecture (Sections 7.3.2 $\sim$ 7.3.4).

7.2 Preliminaries

First, the problem of recommending items from sequences is described. Next, a recent convolutional sequence embedding recommendation model (Caser) is shortly recapitulated along with its limitations. Lastly, we review previous work on sequence-based recommender systems.

7.2.1 Top-N Sequential Recommendation

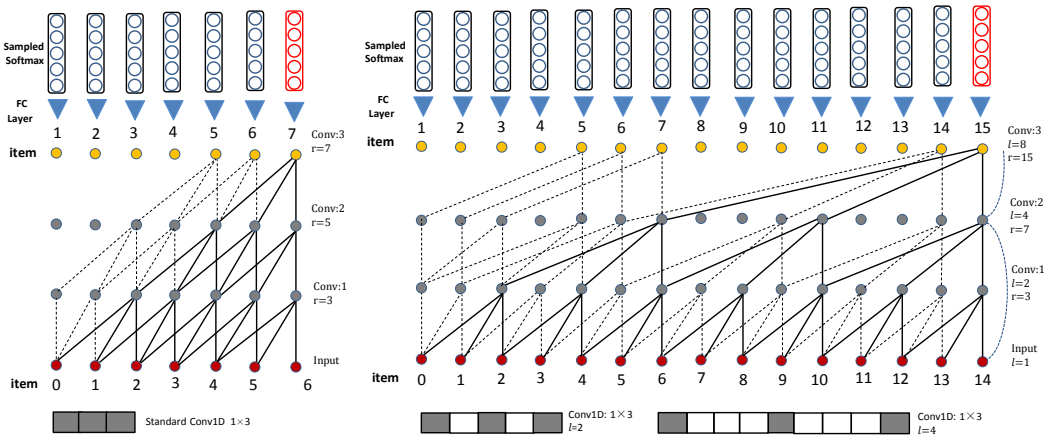
Let $\{x_0, x_1, ..., x_{t-1}, x_t\}$ (interchangeably denoted by $x_{0:t}$) be a clicked item sequence (or a session), where $x_i \in \mathbb{R}$ $(0 \leq i \leq t)$ is the index of the clicked item

out of a total number of $t+1$ items in the sequence. The goal of sequential recommendation is to seek a model such that for a given prefix item sequence, $\mathbf{x} = \{x_0, ..., x_i\}$ $(0 \leq i < t)$, it generates a ranking or classification distribution $\mathbf{y}$ for all candidate items, where $\mathbf{y} = [y_1, ..., y_n] \in \mathbb{R}^n$. y_j can be a score, probability or a rank of item $i+1$ that will occur in this sequence. In practice, we typically make more than one recommendation by choosing the top-N items from $\mathbf{y}$, referred to as the top-N sequential recommendations.

7.2.2 Limitations of Caser

The basic idea of Caser is to embed the previous t items as a $t \times k$ matrix $\mathbf{E}$ by the embedding look-up operation, as shown in Figure 7.1 (a). Each row vector of the matrix corresponds to the latent features of one item. The embedding matrix can be regarded as the "image" of the t items in the k-dimensional latent space. Intuitively, models of various CNNs that are successfully applied in computer vision can be adapted to model the "image" of an item sequence. However, there are two aspects that differentiate sequence modeling from image processing, which makes the use of CNN based models non-straightforward. First, the variable-length item sequences in real-world scenarios produce a large number of "images" of different sizes, where traditional convolutional structures with fix-sized filters may fail. Second, the most effective filters for images, such as 3×3 and 5×5, are not suitable for sequence "images" since these small filters (in terms of row-wise orientation) are not suitable to capture the representations of full-width embedding vectors.

To address the above limitations, filters in Caser slide over full columns of the sequence "image" by large filter. That is, the width of filters is usually the same as the width of the input "images". The height typically varies by sliding windows over $2-5$ items at a time (Figure 7.1 (a)). Filters of different sizes will generate variable-length feature maps after convolution (Figure 7.1 (b)). To ensure that all maps have the same size, the max pooling is performed over each map, which selects only the largest number of each feature map, resulting in a 1×1 map (Figure 7.1 (c)). Finally, these 1×1 maps from all filters are concatenated to form a feature vector, followed by a softmax layer that yields the probabilities of next item (Figure 7.1 (d)). Note that we have omitted the vertical convolution in



(a) Standard CNN (b) Dilated CNN with 'Holes'

Figure 7.2: The proposed generative architecture with 1D standard CNNs (a) and efficient dilated CNNs (b). An example of a standard 1D convolution filter and dilated filters are shown at the bottom of (a) and (b) respectively. The blue dash line is the identity map which exists only for residual block (b) in Fig. 7.3.

Figure 7.1, since it does not solve the major problems discussed below. However, in our experiments, we report final results of Caser with both horizontal and vertical convolutions, although the vertical one results in a very slight accuracy gain.

Based on the above analysis of the convolutions in Caser, one may find that there exist several drawbacks with the current design. First, the max pooling operator has obvious disadvantages. It cannot distinguish whether an important feature in the map occurs just one or multiple times and it ignores the position in which it occurs. The max pooling operator while safely used in image processing (with small pooling filters, e.g., 3×3) may be harmful for modeling long-range sequences (with large filters, e.g., 1×50). Second, the shallow network structure in Caser that suits for only one hidden convolutional layer is likely to fail when modeling complex relations or long-range dependences. The last important disadvantage comes from the generative process of next item, which we will describe in detail in Section 7.3.1.

7.2.3 Related Work

Early work in sequential recommendations mostly rely on the markov chain (Rendle et al., 2010; Cheng et al., 2013; Wang et al., 2015) and feature-based matrix factorization (Rendle, 2012; Chen et al., 2012) approaches. Compared with neural network models, the markov chain based approaches fail to model complicated relations in the sequence data. For example, in *Caser*, the authors pointed out that markov chain approaches failed to model union-level sequential patterns and did not allow skip behaviors in the item sequences. Factorization based approaches such as factorization machines model a sequence by the sum of its item vectors. However, these methods do not consider the order of items and are not specifically invented for sequential recommendations.

Recently, deep learning models have shown state-of-the-art recommendation accuracy in contrast to conventional models. Moreover, RNNs, a class of deep neural networks, have almost dominated the area of sequential recommendations. For example, a Gated Recurrent Unit (GRURec) architecture with a ranking loss was proposed by (Hidasi et al., 2015) for session-based recommendation. In the follow-up papers, various RNN variants have been designed to extend the typical one for different application scenarios, such as by adding personalization (Quadrana et al., 2017), content (Gu et al., 2016) and contextual features (Smirnova and Vasile, 2017), attention mechanism (Cui et al., 2017; Pei et al., 2017; Li et al., 2017) and different ranking loss functions (Hidasi and Karatzoglou, 2017).

By contrast, CNN based sequential recommendation models are more challenging and much less explored because convolutions are not a natural way to capture sequential patterns. To our best knowledge, only two types of sequential recommendation architectures have been proposed to date: the first one by *Caser* is a standard 2D CNN, while the second is a 3D CNN (Tuan and Phuong, 2017) designed to model high-dimensional features. Unlike the aforementioned examples, we propose the use of a stack of 1D CNNs with efficient dilated convolution filters and residual blocks for building the architecture of our generative model.

7.3 Model Design

To address the above limitations, we introduce a new conditional generative model that is formed of a stack of 1D convolution layers. We first focus on the form of the distribution, and then the architectural innovations. Generally, our proposed model is fundamentally different from Caser in several key ways: (1) our probability estimator explicitly models the distribution transition of all individual items at once, rather than the final one, in the sequence; (2) our network has a deep, rather than shallow, structure; (3) our convolutional layers are based on the efficient 1D dilated convolution rather than standard 2D convolution; and (4) pooling layers are not used.

7.3.1 Sequential Generative Modeling

In this section, we propose a generative model directly operating on the sequence of previous items. Our aim is to estimate a distribution over the original item sequences that can be used to tractably compute the likelihood of items and to generate the future items. Let $p(\mathbf{x})$ be the joint distribution of item sequence $\mathbf{x} = \{x_0, ..., x_t\}$. To model $p(\mathbf{x})$, we factorize it as a product of conditional distributions by the chain rule.

$$p(\mathbf{x}) = \prod_{i=1}^{t} p(x_i | x_{0:i-1}, \boldsymbol{\theta}) p(x_0) \tag{7.1}$$

where the value $p(x_i|x_{0:i-1}, \boldsymbol{\theta})$ is the probability of i-th item x_i conditioned on all the previous items $x_{0:i-1}$. A similar setup has been adopted by NADE (Larochelle and Murray, 2011), PixelRNN/CNN (Oord et al., 2016b; van den Oord et al., 2016) in biological and image domains. To model the highly complicated function in Eq. (7.1), an expressive sequence model is necessary.

Owing to the ability of neural networks in modeling complex nonlinear relations, in this chapter we model the conditional distributions by a stack of 1D convolutional networks. To be more specific, the network receives $x_{0:t-1}$ as the input and outputs distributions over possible $x_{1:t}$, where the distribution of x_t is our final expectation. For example, as illustrated in Figure 7.2, the output distribution of x_{15} is determined by $x_{0:14}$, while $x_{14}$is determined by $x_{0:13}$. It is worth noting that in previous sequential recommendation literatures, such as Caser,

GRURec and Li et al. (2017); Quadrana et al. (2017); Tuan and Phuong (2017); Tan et al. (2016), they only model a single conditional distribution $p(x_i|x_{0:i-1}, \boldsymbol{\theta})$ rather than all conditional probabilities $\prod_{i=1}^{t} p(x_i|x_{0:i-1}, \boldsymbol{\theta})p(x_0)$. Within the context of the above example, assuming $\{x_0, ..., x_{14}\}$ is given, models like Caser only estimate the probability distribution (i.e., softmax) of the next item x_{15} (also see Figure 7.1 (d)), while our generative method estimates the distributions of all individual items in $\{x_1, ..., x_{15}\}$. The comparison of the generating process is shown below.

$$\begin{aligned} Caser/GRURec &: \underbrace{\{x_0, x_1, ..., x_{14}\}}_{input} \Rightarrow \underbrace{x_{15}}_{output} \\ Ours &: \underbrace{\{x_0, x_1, ..., x_{14}\}}_{input} \Rightarrow \underbrace{\{x_1, x_2, ..., x_{15}\}}_{output} \end{aligned} \tag{7.2}$$

where $\Rightarrow$ denotes 'predict'. Clearly, our proposed model is more effective in capturing the set of all sequence relations, whereas Caser and GRURec fail to explicitly model the internal sequence features between $\{x_0, ..., x_{14}\}$. In practice, to address the drawback, such models will typically generate a number of sub-sequences (or sub-sessions) for training by means of data augmentation techniques (Tan et al., 2016) (e.g., padding, splitting or shifting the input sequence), such as shown in Eq. (7.3) (see (Tuan and Phuong, 2017; Li et al., 2017; Quadrana et al., 2017; Tang and Wang, 2018)).

$$\begin{aligned} &\text{Caser/GRURec sub-session} - 1 : \{x_{-1}, x_0, ..., x_{13}\} \Rightarrow x_{14} \\ &\text{Caser/GRURec sub-session} - 2 : \{x_{-1}, x_{-1}, ..., x_{12}\} \Rightarrow x_{13} \\ &\qquad\qquad ..... \\ &\text{Caser/GRURec sub-session} - 12 : \{x_{-1}, x_{-1}, ..., x_2\} \Rightarrow x_3 \end{aligned} \tag{7.3}$$

While effective, the above approach to generate sub-session cannot guarantee the optimal results due to the separate optimization for each sub-session. In addition, optimizing these sub-sessions separately will result in corresponding computational costs. Detailed comparison with empirical results has also been reported in our experimental sections.

In summary, our generative model has three properties: (1) same as Caser and GRURec, it is also autoregressive in the target item (as opposed to GANs (Goodfellow et al., 2014)); (2) different from Caser and GRURec, it is capable

of explicitly modeling the complete set of sequential dependencies in a natural way without resorting to more additional sub-sequences; and (3) it is sensitive to the ordering of the items in both input and output. An additional advantage is that pooling layers are not used in our architecture, and thus the output of the convolutional layer is guaranteed to have the same spatial dimensions as the input. The unchanged dimensionality makes layer-wise stacking and residual learning much easier, which is detailed later.

7.3.2 Network Architecture

The network architecture is shown in Figure 7.2. We will refer to a dilated convolution with a dilation factor l as l-dilated convolution. As can be seen, compared with the standard CNN that linearly increases the receptive field, the dilated CNN has a much larger receptive field by the same stacks without introducing more parameters.

7.3.2.1 Embedding Look-up

Given an item sequence $\{x_0, ..., x_t\}$, the model retrieves each of the first t items $\{x_0, ..., x_{t-1}\}$ via a look-up table, and stacks these item embeddings together. Assuming the embedding dimension is $2k$, where k will be set as the number of inner channels in the convolutional network. This results in a matrix of size $t \times 2k$. Caser on the other hand treats the input matrix as a 2D "image" during convolution, while the proposed architecture learns the sequential representation by 1D convolutional filters, which we will describe later.

7.3.2.2 Dilation

Instead of using standard convolutions, we apply the dilated ones to construct the proposed generative model. Dilated convolutions were first proposed in Yu and Koltun (2015), which enlarge the receptive field of images without increasing the filter size. The basic idea of dilation is to apply the convolutional filter over a field larger than its original length by dilating it with zeros. As such, it is more efficient since it utilizes fewer parameters. For this reason, a dilated filter is also referred to as a holed or sparse filter. Another benefit is that dilated convolution can

preserve the spatial dimensions of the input, which makes the stacking operation much easier for both convolutional layers and residual structures.

Figure 7.2 shows the network comparison between the standard convolution and dilated convolutions with the proposed sequential generative model. The dilation factor in (b) are $1, 2, 4$ and 8. To describe the network architecture, we denote receptive field, j-th convolutional layer, channel and dilation as r, F_j, C and l respectively. By setting the width of convolutional filter f as 3, we can see that the dilated convolutions (Fig. 7.2 (b)) allow for exponential increase in the size of receptive fields ($r = 2^{j+1} - 1$), while the same stacking structure for the standard convolution (Fig. 7.2 (a)) has only linear receptive fields ($r = 2j + 1$). Formally, with dilation l, the filter window from location i is given as

$$\begin{bmatrix} x_i & x_{i+l} & x_{i+2l} & \dots & x_{i+(f-1)\cdot l} \end{bmatrix}$$

Clearly, the dilated convolutional structure is more effective to model long-range item sequences, and thus more efficient without using larger filters or becoming deeper. In practice, to further increase the model capacity and receptive fields, one just need to simply repeat the architecture in Fig. 7.2 multiple times by stacking, e.g., $1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8$.

7.3.2.3 One-dimensional Transformation

Although our dilated convolution operator depends on the 2D input matrix $\boldsymbol{E}$, the proposed network architecture is actually composed of all 1D convolutional layers. To model the 2D embedding input, we perform a reshape operation, which serves as a prerequisite for performing 1D convolution. Specifically, the 2D matrix $\boldsymbol{E}$ is reshaped from $t \times 2k$ to a 3D tensor $\boldsymbol{T}$ of size $1 \times t \times 2k$, where $2k$ is treated as the "image" channel rather than the width of the standard convolution filter in *Caser*. Figure 7.3 (b) illustrates the reshaping process.

7.3.3 Residual Learning

Although increasing the depth of network layers can help obtain higher-level feature representations, it also easily results in the vanishing gradient issue, which makes the learning process much harder. To address the degradation problem, residual learning (He et al., 2016a,b) has been introduced for deep networks.

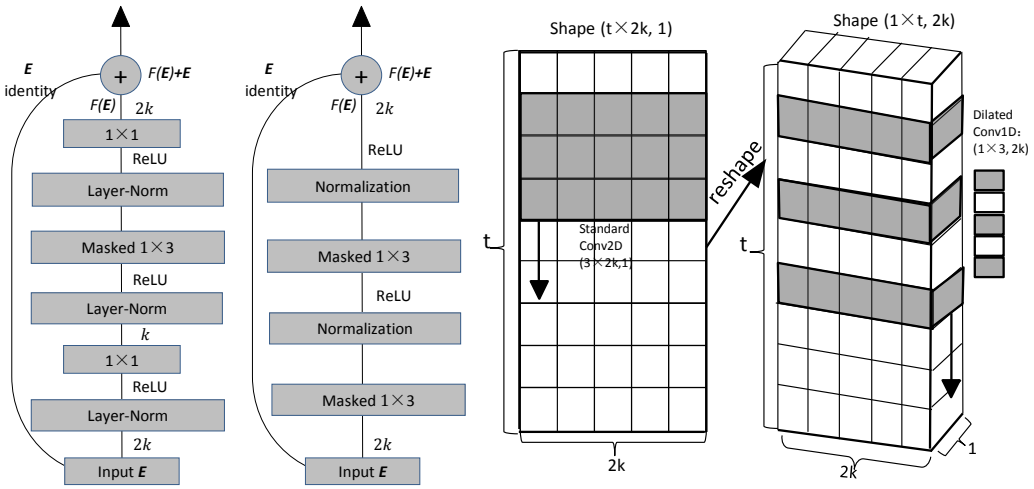


Figure 7.3: Dilated residual blocks (a), (b) and one-dimensional transformation (c). (c) shows the transformation from the 2D filter ($C = 1$)(left) to the 1D 2-dilated filter ($C = 2k$) (right); the vertical black arrows represent the direction of the sliding convolution. In this work, the default stride for the dilated convolution is 1. Note the reshape operation in (b) is performed before each convolution in (a) and (b) (i.e., 1×1 and masked 1×3), which is then followed by a reshape back step after convolution.

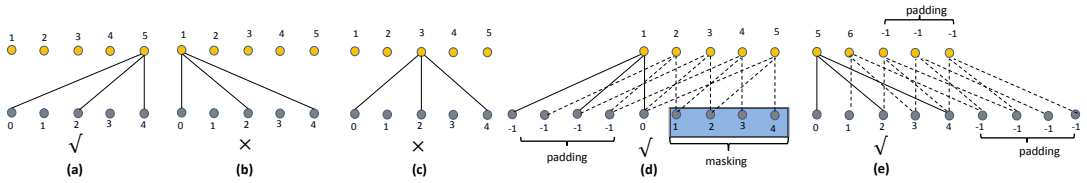


Figure 7.4: The future item can only be determined by the past ones according to Eq. (7.1). (a) (d) and (e) show the correct convolution process, while (b) and (c) are wrong. E.g., in (d), items of $\{1, 2, 3, 4\}$ are masked when predicting 1, which can be technically implemented by padding.

While residual learning has achieved huge success in the domain of computer vision, to our knowledge it has not appeared in the recommender system literature.

The basic idea of residual learning is to stack multiple convolutional layers together as a block and then employ a skip connection scheme that passes the previous layers's feature information to its posterior layer. The skip connection

scheme allows to explicitly fit the residual mapping rather than the original identity mapping, which can maintain the input information and thus enlarge the propagated gradients. Formally, denoting the desired mapping as $H(\boldsymbol{E})$, we let the residual block fit another mapping of $F(\boldsymbol{E}) = H(\boldsymbol{E}) - \boldsymbol{E}$. The desired mapping now is recast into $F(\boldsymbol{E}) + \boldsymbol{E}$ by element-wise addition (assuming that $F(\boldsymbol{E})$ and $\boldsymbol{E}$ are of the same dimension). As has been evidenced in He et al. (2016a), optimizing the residual mapping $F(\boldsymbol{E})$ is much easier than the original, unreferenced mapping $H(\boldsymbol{E})$. Inspired by He et al. (2016b); Kalchbrenner et al. (2016), we introduce two alternative residual modules in Fig. 7.3 (a) and (b).

In (a), we wrap each dilated convolutional layer by a residual block, while in (b) we wrap every two dilated layers by a different residual block. Note with the design of block (b), the input layer and the second convolutional layer should be connected by skip connection (i.e., the blue dash line in Fig. 7.2). Specifically, each block is made up of the normalization, activation (e.g., ReLU Nair and Hinton (2010)), convolutional layers and a skip connection in a specific order. In this work we adopt the state-of-the-art layer normalization Ba et al. (2016) before each activation layer, as it is well suited to sequence processing in contrast with batch normalization Ioffe and Szegedy (2015).

Regarding the properties of the two residual networks, the residual block in (a) consists of 3 convolution filters: one dilated filter of size 1×3 and two regular filters of size 1×1. The 1×1 filters are introduced to change the size of C so as to reduce the parameters to be learned by the 1×3 kernel. The first 1×1 filter (close to input $\boldsymbol{E}$ in Fig. 7.3 (a)) is to change C from $2k$ to k, while the second 1×1 filter does the opposite transformation in order to maintain the spatial dimensions for the next stacking operation. To show the effectiveness of the 1×1 filters in (a), we compute the number of parameters in both (a) and (b). For simplicity, we omit the activation and normalization layers. As we can see, the number of parameters for the 1×3 filter is $1 \times 3 \times 2k \times 2k = 12k^2$ (i.e., in (b)) without the 1×1 filters. While in (a), the number of parameters to be learned is $1 \times 1 \times 2k \times k + 1 \times 3 \times k \times k + 1 \times 1 \times k \times 2k = 7k^2$. Although the simple residual module in (b) has more parameters, we empirically find that it is a bit more effective than (a) when the sequential relations are relatively weak.

Table 7.1: Session statistics of all data sets. MUSIC_M5 denotes MUSIC_M with maximum session size of 5. The same applies to MUSIC_L. 'M' denotes 1 million.

DATA	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
RAW-SESSIONS	0.14M	0.61M	2.14M	1.07M	0.53M	0.21M	0.11M
SUB-SESSIONS-T	0.07M	0.31M	1.07M	3.21M	4.28M	4.91M	5.10M

The residual mapping $F(\boldsymbol{E}, \{W_i\})$ in (a) and (b) is formulated as:

$$F(\boldsymbol{E}, \{W_i\}) = \begin{cases} W_3(\sigma(\psi(W_2(\sigma(\psi(W_1(\sigma(\psi(\boldsymbol{E})))))))))) & \text{Fig.7.3 (a)} \\ \sigma(\psi(W_4'(\sigma(\psi(W_2'(\boldsymbol{E})))))) & \text{Fig.7.3 (b)} \end{cases} \tag{7.4}$$

where σ and ψ denote ReLU and layer-normalization, W_1 and W_3 denote the convolution weight function of standard 1×1 convolutions, while W_2, W_2' and W_4' denote the weight function of l-dilated convolution filter with size of 1×3. Note that bias terms are omitted for simplifying notations.

7.3.3.1 Masking

We propose a masking strategy for the 1D dilated convolution to prevent the network from seeing the future items in the output sequence. Specifically, when predicting $p(x_i|x_{0:i-1})$, the convolution filters are not allowed to make use of the information from $x_{i:t}$. Figure 7.4 shows several different ways to perform convolution. As shown, the masking operation can be implemented either by padding the input sequence in (d) or shifting the output sequence by a few timesteps in (e). The padding method in (e) is very likely to result in information loss in a sequence, particularly for short sequences. Hence in this work, we apply the padding strategy in (d) with the padding size of $(f-1) * l$.

7.3.4 Final Layer, Network Training and Generating

As mentioned, the matrix in the last layer of the convolution architecture (see Fig. 7.2), denoted by $\boldsymbol{E}^o$, preserves the same dimensional size of the input $\boldsymbol{E}$, i.e., $\boldsymbol{E}^o \in \mathbb{R}^{t \times 2k}$. However, the output should be a matrix or tensor that contains probability distributions of all items in the output sequence $x_{1:t}$, where the probability distribution of x_t is the desired one that generates top-N predictions. To do this, we can simply use one more convolutional layer on top of the last convolutional layer in Fig. 7.2 with filter of size $1 \times 1 \times 2k \times n$, where n is the

Table 7.2: Accuracy comparison. The upper, middle and below tables are MRR@5, HR@5 and NDCG@5 respectively.

	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
MostPop	0.0050	0.0024	0.0006	0.0007	0.0008	0.0007	0.0007
GRURec	0.1645	0.3019	0.2184	0.2124	0.2327	0.2067	0.2086
Caser	0.1523	0.2920	0.2207	0.2214	0.1947	0.2060	0.2080
NextItNet	**0.1715**	**0.3133**	**0.2327**	**0.2596**	**0.2748**	**0.2735**	**0.2583**
MostPop	0.0151	0.0054	0.0014	0.0016	0.0016	0.0016	0.0016
GRURec	0.2773	0.3610	0.2626	0.2660	0.2694	0.2589	0.2593
Caser	0.2389	0.3368	0.2443	0.2631	0.2433	0.2572	0.2588
NextItNet	**0.2871**	**0.3754**	**0.2695**	**0.3014**	**0.3166**	**0.3218**	**0.3067**
MostPop	0.0075	0.0031	0.0008	0.0009	0.0010	0.0009	0.0009
GRURec	0.1923	0.3166	0.2294	0.2258	0.2419	0.2197	0.2212
Caser	0.1738	0.3032	0.2267	0.2318	0.2068	0.2188	0.2207
NextItNet	**0.2001**	**0.3288**	**0.2419**	**0.2700**	**0.2853**	**0.2855**	**0.2704**

MostPop returns the most popular item respectively. Regarding the setup of our model, we use two-hidden-layer convolution structure with dilation factor $1, 2, 4$ for the first four data sets (i.e., YOO, MUSIC_M5 , MUSIC_L5 and MUSIC_L10), while for the last three long-range sequence data sets, we use $1, 2, 4, 8, 1, 2, 4, 8$, to obtain above results.

number of items. Following the procedure of one-dimensional transformation in Fig. 7.3 (c), we obtain the expected ouput matrix $\boldsymbol{E}^p \in \mathbb{R}^{t \times n}$, where each row vector after the softmax operation represents the categorical distribution over x_i $(0 < i \leq t)$.

The aim of optimization is to maximize the log-likelihood of the training data w.r.t. $\boldsymbol{\theta}$. Clearly, maximizing $\log p(\mathbf{x})$ is mathematically equivalent to minimizing the *sum* of the binary cross-entropy loss for each item in $x_{1:t}$. For practical recommender systems with tens of millions items, the negative sampling strategy such as sampled softmax (Jean et al., 2014) can be applied to bypasses the generation of full softmax distributions, where the 1×1 convolutional layer is replaced by a fully-connected (FC) layer with weight matrix $\boldsymbol{E}^g \in \mathbb{R}^{2k \times n}$. The recommendation accuracy by the negative sampling strategies is nearly identical with the full softmax method with properly tuned sampling size.

For comparison purpose, we only predict the next one item in our evaluation, and then stop the generating process. Nevertheless, the model is able to generate a sequence of items simply by feeding the predicted one item (or sequence) into the network to predict the next one, and thus the prediction at the generating phrase is *sequential*. This matches most real-world recommendation scenarios, where the next action is followed when the current one has been observed. But at both training and evaluation phases, the conditional predictions for all timesteps can be made *in parallel*, because the complete sequence of input items $\mathbf{x}$ is already available.

7.4 Experiments

In this section we detail our experiments, report results for several data sets, and compare our model (called NextItNet) with the well-known RNN-based model GRURec[1] (Hidasi et al., 2015; Tan et al., 2016) and the state-of-the-art CNN-based model Caser[2] (Tang and Wang, 2018). Note that (1) since the main contributions in this chapter do not focus on combining various features, we omit the comparison with content- or context-based sequential recommendation models, such as the 3D CNN recommender (Tuan and Phuong, 2017) and other RNN variants (Gu et al., 2016; Smirnova and Vasile, 2017; Li et al., 2017; Quadrana et al., 2017); (2) the GRURec baseline could be roughly regarded as the state-of-the-art Improved GRURec (Tan et al., 2016) when dealing with the long-range session data sets because our main data augmentation technique for the two baseline models follows the way of Improved GRURec.

7.4.1 Datasets and Experiment Setup

7.4.1.1 Datasets and Preprocessing

The first data set 'Yoochoose-buys' (YOO for short) is chosen from the RecSys Challenge 2015[3], which contains buying and clicking events. We only keep the buying data. To avoid noise data, we filter out sessions of length shorter than 3. To fairly compare the capacity of NextItNet in modeling sequences, we do not consider additional contexts in this chapter, although Yoo contains item price and quality information. We find that in the processed Yoo data 96% sessions have a length shorter than 10, and we remove the 4% longer sessions and refer it as a short-range sequential data.

The remaining data sets are extracted from Last.fm[4]: one medium-size (MUSIC_M) and one large-scale (MUSIC_L) collection by randomly drawing 20,000 and 200,000 songs respectively. In the Last.fm data set, we observe that most users listen to music several hundred times a week, and some even listen to more than one hundred songs within a day. Hence, we are able to test our model in

[1] https://github.com/hidasib/GRU4Rec
[2] https://github.com/graytowne/caser
[3] http://2015.recsyschallenge.com/challenge.html
[4] http://www.dtic.upf.edu/ ocelma/MusicRecommendationDataset/lastfm-1K.html

Table 7.3: Accuracy comparison. The upper, middle and below tables are MRR@20, HR@20 and NDCG@20 respectively.

	YOO	MUSIC_M5	MUSIC_L5	MUSIC_L10	MUSIC_L20	MUSIC_L50	MUSIC_L100
MostPop	0.0090	0.0036	0.0009	0.0010	0.0011	0.0011	0.0011
GRURec	0.1839	0.3103	0.2242	0.2203	0.2374	0.2151	0.2162
Caser	0.1660	0.2979	0.2234	0.2268	0.2017	0.2133	0.2153
NextItNet	**0.1901**	**0.3223**	**0.2375**	**0.2669**	**0.2815**	**0.2794**	**0.2650**
MostPop	0.0590	0.0180	0.0052	0.0053	0.0056	0.0056	0.0056
GRURec	0.4603	0.4435	0.3197	0.3434	0.3158	0.3406	0.3336
Caser	0.3714	0.3937	0.2703	0.3150	0.3110	0.3273	0.3298
NextItNet	**0.4645**	**0.4626**	**0.3159**	**0.3709**	**0.3814**	**0.3789**	**0.3731**
MostPop	0.0195	0.0066	0.0018	0.0019	0.0021	0.0020	0.0020
GRURec	0.2460	0.3405	0.2460	0.2481	0.2553	0.2433	0.2427
Caser	0.2122	0.3197	0.2342	0.2469	0.2265	0.2392	0.2412
NextItNet	**0.2519**	**0.3542**	**0.2554**	**0.2904**	**0.3041**	**0.3021**	**0.2895**

both short- and long-range sequences by cutting up these long-range listening sessions. In MUSIC_L, we define the maximum session length t as 5, 10, 20, 50 and 100, and then extract every t successive items as our input sequences. This is done by sliding a window of both size and stride of t over the whole data. We ignore sessions in which the time span between the last two items is longer than 2 hours. In this way, we create 5 data sets, referred to as RAW-SESSIONS. We randomly split these RAW-SESSIONS data into training (50%), validation (5%), and testing (45%) sets.

In our evaluation, we observe that the performance of Caser and GRURec degrades significantly for very long sequence inputs, such as when $t = 20$, 50 and 100. As mentioned before, for example, when setting $t = 50$, Caser and GRURec will predict x_{49} by using $x_{0:48}$, but without explicitly modeling the item inter-dependencies between x_0 and x_{48}. To remedy this defect, when $t > 5$, we follow the common approach (Tan et al., 2016; Li et al., 2017) by manually creating additional sessions from the training sets of RAW-SESSIONS so that Caser and GRURec can leverage the full dependency to a large extent. Still setting $t = 50$, one training session will then produce 45 more sub-sessions by padding the beginning and removing the end indices, referred to as SUB-SESSIONS. The example of the 45 sub-sessions are given as follows: $\{x_{-1}, x_0, x_1, ..., x_{48}\}$, $\{x_{-1}, x_{-1}, x_0, ..., x_{47}\}$,..., $\{x_{-1}, x_{-1}, x_{-1}, ..., x_4\}$. In the evaluation, we find there are no further improvements even we produce more fine-grained sub-sessions, such as $\{x_{-1}, x_{-1}, x_{-1}, ..., x_3\}$. Regarding MUSIC_M, we only show the results when $t = 5$ due to the similar trend in MUSIC_L. We show the statistics of

Table 7.4: Effects of sub-session in terms of MRR@5. The upper, middle and below tables represent GRU, Caser and NextItNet respectively. "10", "20", "50" and "100" the session length. All high parameters are fixed.

Sub-session	10	20	50	100
Without	0.1985	0.1645	0.1185	0.0746
With	0.2124	0.2327	0.2067	0.2086
Without	0.1571	0.1012	0.0216	0.0084
With	0.2214	0.1947	0.2060	0.2080
Without	0.2596	0.2748	0.2735	0.2583

Table 7.5: Effects (MRR@5) of increasing embedding size. The upper and below tables are MUSIC_M5 and MUSIC_L100 respectively.

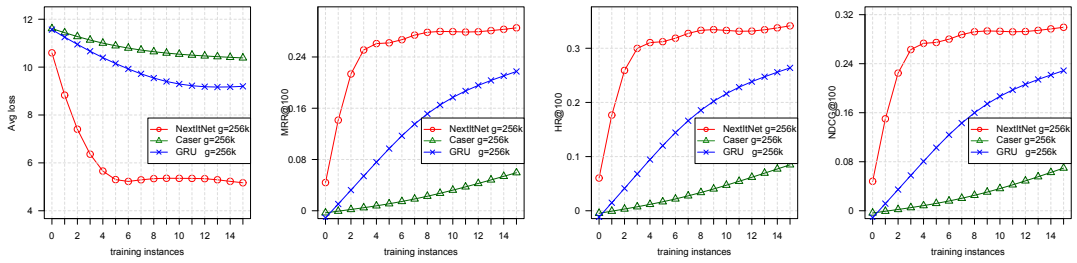
$2k$	16	32	64	128
GRURec	0.2786	0.2955	0.3019	0.3001
Caser	0.2855	0.2982	0.2979	0.2958
NextItNet	0.2793	0.3063	0.3133	0.3183
GRURec	0.1523	0.1826	0.2086	0.2043
Caser	0.0643	0.1129	0.2080	0.2339
NextItNet	0.1668	0.2289	0.2583	0.2520

Note that all three models use $2k$ as the embedding size, where k in our model is the number of inner channels.

RAW-SESSIONS & training data of SUB-SESSIONS (i.e., SUB-SESSIONS-T) in Table 7.1.

7.4.1.2 Hyper-parameter Settings

All models were trained on GPUs using Tensorflow. From the different gradient descent optimizers tried, Adam (Kingma and Ba, 2014) gave the best convergence performance and was used for all evaluations. The learning rates and batch sizes of baseline methods were manually set according to performance in validation sets. For all data sets, NextItNet used the learning rate of 0.001 and batch size of 32. Embedding size $2k$ is set to 64 for all models without special mention. In addition, for comparison, we report all results with residual block (a) and full softmax. We have validated the performance of results block (b) separately.



(a) Loss (b) MRR@5 (c) HR@5 (d) NDCG@5

Figure 7.5: Convergence behaviors of MUSIC_L100. GRU is short for GRURec. $g = 256$k means the number of training sequences (or sessions) of one unit in x-axis is 256k. Note that (1) to speed up the evaluation, all of the convergence tests are performed on the first 1024 sessions in the testing set, which also applies to Fig. 7.6; (2) clearly, GRU and Caser have not converged in above figures.

7.4.1.3 Evaluation Protocols

We reported the evaluated results by three popular top-N metrics, namely MRR@N (Mean Reciprocal Rank) (Hidasi et al., 2015) and HR@N (Hit Ratio) (He et al., 2016c) and NDCG@N (He et al., 2016c) (Normalized Discounted Cumulative Gain). MRR and NDCG take the rank of the item into account, which is important in settings where the order of recommendations matters, while HR@N does not consider the actual rank of the item as long as it is amongst the top-N. For saving spaces, we have omitted the detailed formulas. N is set to 5 and 20 for comparison. We evaluate the prediction accuracy of the last (i.e., next) item of each sequence in the testing set, similarly to Hidasi and Karatzoglou (2017); Tang and Wang (2018).

7.4.2 Results Summary

Overall performance results of all methods are illustrated in Table 7.2 and 7.3, which clearly show that the neural network models (i.e., Caser, GRURec and our model) obtain very promising accuracy in the top-N sequential recommendation task. For example, in MUSIC_M5, the three neural models perform more than 120 times better on MRR@5 than MostPop, which is a widely used recommendation benchmark. The best MRR@20 result we have achieved by NextItNet is 0.3223 in this data set, which roughly means that the desired item is ranked on position 3 in average among the 20,000 candidate items. We then find that

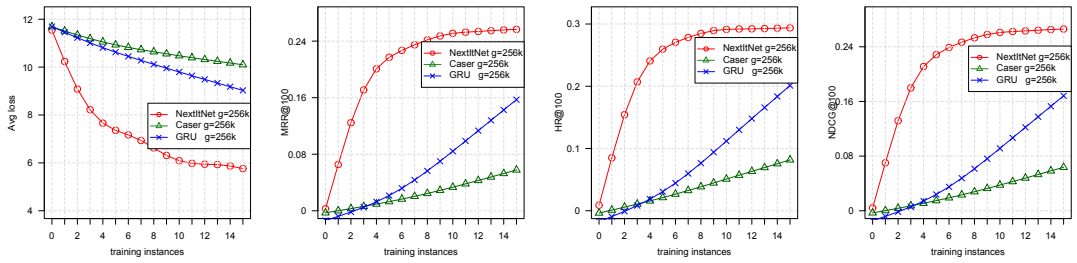
Table 7.6: Effects of the residual block in terms of MRR@5. "Without" means no skip connection. "M5", "L5", "L10" and "L50" denote MUSIC_M5 , MUSIC_L5 , MUSIC_L10 and MUSIC_L50 respectively. All high parameters are fixed.

DATA	M5	L5	L10	L50
Without	0.2968	0.2146	0.2292	0.2432
With	0.3300	0.2455	0.2645	0.2760

among these neural network based models, NextItNet largely outperforms Caser & GRURec. We believe there are several reasons contributing to the state-of-the-art results. First, as highlighted in Section 7.3.1, NextItNet takes full advantage of the complete sequential information instead of just the sequence information between the last item and the previous sequence. This can be easily verified in Table 7.4, where we observe that Caser & GRURec without subsession perform extremely worse with long sessions. Particularly, Caser performs much worse than GRURec. The reason is because compared with a regular CNN model[1], RNN has an inherent advantage in modeling long-range sequences. More importantly, the poor performance of Caser is likely to be attributed to the max pooling operation, which seriously hurts its performance when modeling long-range sequences. In addition, even with sub-session Caser & GRURec still show significantly worse results than NextItNet because the separate optimization of each sub-session is clearly suboptimal compared with leveraging full sessions by NextItNet[2]. Second, unlike Caser, NextItNet has no pooling layers, although it is also a CNN based model. As a result, NextItNet preserves the whole spatial resolution of the original embedding matrix $\boldsymbol{E}$ without any information lost. The third advantage may be that *NextItNet* can support deeper layers by using residual learning, which better suits for modeling complicated relations and long-range dependencies. We have separately validated the performance of residual block in Fig. 7.3 (b) and showed the results in Table 7.6. It can be observed that the performance of *NextItNet* can be significantly improved by the residual block design. We also report results with varying embedding size in Table 7.5, which shows that a relatively larger embedding size will increase the prediction accuracy. Empirically, the model performs well enough when $2k$ is between 64 and 128.

[1] In fact, Caser is not a real sequential model because the max pooling operation does not retain sequential relations of items.

[2] Note that NextItNet is also able to model sub-sessions similarly as Caser & GRURec when necessary.



(a) Loss (b) MRR@5 (c) HR@5 (d) NDCG@5

Figure 7.6: Convergence behaviors of MUSIC_L20.

Table 7.7: Overall training time (mins).

Model	GRURec	Caser	NextItNet
MUSIC_L5	66	59	54
MUSIC_L20	282	191	76
MUSIC_L100	586	288	150

In addition to the advantage of recommendation accuracy, we have also evaluated the efficiency of NextItNet in Table 7.7. All methods are compared by outputting the full softmax distribution. First, it can be seen that NextItNet and Caser requires less training time than GRURec in all three data sets. This reason that CNN-based models can be trained much faster is due to the full parallel mechanism of convolutions. Clearly, the training speed advantage of CNN models are more preferred by modern parallel computing systems. Second, it shows that NextItNet achieves further improvements in training time compared with Caser. The faster training speed is mainly because NextItNet leverages the complete sequential information during training and then converges much faster by less training epochs. To better understand of the convergence behaviours, we have shown them in Fig. 7.5 and 7.6. As can be seen, our model with the same number of training sessions is converging faster (a) and better (b, c, d) than Caser and GRURec. This confirms our claim in Section 7.3.1 since Caser and GRURec cannot make full use of the internal sequential information in the session. The faster and better convergence behaviors indicate the effectiveness of the proposed generative architecture.

7.5 Chapter Summary

In this chapter, we presented a simple, efficient and effective convolutional generative model for session-based top-N item recommendations. The proposed model combines masked filters with 1D dilated convolutions to increase the receptive fields, which is very important to model the long-range dependencies. In addition, we have applied residual learning to enable training of much deeper networks. We have shown that our model can greatly outperform state-of-the-arts in real-world session-based recommendation tasks. The proposed model can serve as a generic method for modeling both short- and long-range session-based recommendation data.

For comparison purposes, we have not considered additional contexts in either our model or baselines. However, our model is flexible to incorporate various context information. For example, if we know the user identity u and location p, the distribution in Eq. (7.1) can be modified as follows to incorporate these information.

$$p(\mathbf{x}) = \prod_{i=1}^{t} p(x_i | x_{0:i-1}, \mathbf{u}, \boldsymbol{P}, \boldsymbol{\theta}) p(x_0) \tag{7.5}$$

where we can combine $\boldsymbol{E}$ (before convolution) or $\boldsymbol{E^o}$ (after convolution) with the user embedding vector $\mathbf{u}$ and location matrix $\boldsymbol{P}$ by element-wise operations, such as multiplication, addition or concatenation. We leave the evaluation for future work.

Part V
Conclusion

In the previous chapters of this thesis, we have studied the problem of item recommendation using implicit feedback from two critical aspects: recommendation accuracy and efficiency. In terms of recommendation accuracy, we have presented LambdaFM, BoostFM for top-N context-aware recommendations, and GeoBPR for location based recommendation. In terms of recommendation efficiency, we have proposed f_{BGD}, which has the same magnitude computational complexity with negative sampling based methods although it takes advantage of all training examples from implicit feedback. All these models showed superb performance in respective recommendation tasks. The state-of-the-art performance may attribute to the smart way of leveraging negative examples. In recent two years, deep learning models have achieved huge success in various research fields. The final contribution of this thesis is to introduce a deep learning based sequential recommendation model, which generate recommendation by a probability output rather than by the prediction scores like above shallow embedding models.

Chapter 8

Conclusions and Future Work

8.1 Contribution Summary

In this thesis, we have studied the item recommendation task from implicit feedback. Our focus is on both recommendation accuracy and efficiency in real-world recommender systems.

First, regarding the recommendation accuracy, we found that most previous works perform uniform sampling from large-scale unobserved implicit feedback. However, by an insightful analysis from the ranking perspective, we observed that most randomly sampled negative examples are not good examples because they fail to improve the top-N ranking measures (e.g., NDCG). By contrast, negative examples that are ranked at the top positions of the rank list, referred to as good (or informative) examples, have more contributions for improving the evaluation metrics. The idea inspires two research work, i.e., LambdaFM (Yuan et al., 2016b,c) in Chapter 2 and BoostFM (Yuan et al., 2017) in Chapter 3. Both models are designed and evaluated for the top-N context-aware recommendation tasks from implicit feedback. The intuition and sampling strategies are also consistent, i.e., oversampling informative unobserved examples as negative for model training. Regarding LambdaFM, we have designed three lambda-motivated negative samplers, including a static sampler and two dynamic samplers. All the three negative samplers show better performance than the uniform sampler in original cross-entropy based pairwise ranking factorization machines (PRFM). In addition, we have successfully verified our LambdaFM by a variety of pairwise loss functions, including hinge loss, exponential loss and fidelity loss. For BoostFM, we only studied the static sampler for the component recommenders by consid-

ering the fact that the two dynamic samplers are much slower than the static sampler. The main difference of the two models is that LambdaFM is a single model, whereas BoostFM is a boosting-based ensemble model. LambdaFM can be seen as a component recommender in BoostFM. BoostFM is able to recover LambdaFM when there is only one boosting round. In terms of recommendation accuracy, both LambdaFM and BoostFM have shown significantly better performance than a bunch of state-of-the-art recommendation models. Particularly, LambdaFM is regarded as a powerful baseline in several recommendation literature, and shows largely better performance than other baseline models (Wang et al., 2017b; Zhao et al., 2017). The performance improvements of both models are attributed to the fact that they are able to more effectively leverage informative negative examples than many baseline models. In contrast with LambdaFM, BoostFM is empirically observed with better accuracy when they are trained with the same sampler. This is because BoostFM is built on many individual component recommenders, which can help reduce the variance during training. The extensive study on the performance of LambdaFM and BoostFM empirically validates our thesis **statement (1)** and **(2)** in Chapter 1.

Regarding the GeoBPR model (Yuan et al., 2016d) in Chapter 5, it is designed by combining geographical preference and pairwise ranking model, referred to as a co-pairwise ranking model. In this work, our main contribution is to improve the original pairwise ranking assumption by taking into account of geographical preference. To our knowledge, it is the first work to change the pairwise ranking assumption by injecting users' geographical preference. Although the model derivation is derived from the ranking perspective, its implementation is still by means of negative sampling strategy — sampling more unobserved POIs that have higher geographical preference. The intuition of GeoBPR is similar to that of LambdaFM and BoostFM as all the three models construct training pairs by leveraging informative unobserved examples. However, compared with LambdaFM and BoostFM, GeoBPR is only suitable for location recommendation. Hence, the work of GeoBPR can also support our thesis **statement (1)** and **(2)**, where nearby unobserved POIs are regarded as informative negative examples.

Second, regarding recommendation efficiency, we have presented the f_{BGD}

model (Yuan et al., 2018b) in Chapter 6. The main motivation of f_{BGD} is that negative sampling based methods are sensitive to the sampling distribution and sampling size of unobserved examples. Moreover, sampling a fraction of unobserved data as negative for training may ignore other important examples. Inspired by this, we designed a full batch gradient descent based recommendation model that leverages all unobserved examples without any sampling. However, it is known that the standard batch gradient descent method suffers from expensive computation cost and is generally infeasible in practice. To solve the efficiency issue, we perform a series of optimized mathematical derivation on the objective function. f_{BGD} is accelerated by several magnitudes, by which its time complexity can reach the same level as the negative sampling based SGD method. Another contribution is that we observe that the standard batch gradient descent method suffers from gradient instability issues and performs poorly on recommendation datasets when there are more than two input features. We then provide an insightful theoretical analysis on this issue and solve it by an intuitive solution. During the evaluation, we observe that f_{BGD} largely outperforms a variety of negative sampling based recommendation models with the same level of computational costs. The experimental results validate our thesis **statement (3)** and **(4)** in Chapter 1.

The last contribution of this thesis is that we introduced NextItNet (Yuan et al., 2018a), a session-based recommendation model which is based on the one-dimensional convolutional neural networks (CNN) rather than the popular recurrent neural networks (RNN). The main motivation is that RNN-based models depend on the hidden state of the entire past that prevents parallel computation. By contrast, training CNNs does not depend on the computations of the previous time step and therefore allows parallelization over every element in a sequence. It is worth noticing that the session-based recommendation task is also based on implicit feedback. However, different from previous models, NextItNet is a generating model that estimates the distribution of the output sequence rather than calculating the relevance scores that are used to generate top-N items list. In the implementation of NextItNet, we output the softmax distribution of all items in the list without resorting to sampling methods. In our evaluation, we show that NextItNet cannot only generate more accurate recommendations than

baselines, but also are much faster than them.

8.2 Future Work

In this section, we have figured out several promising future work regarding implicit feedback based item recommendation.

- Dynamic negative sampling for deep embedding models: The dynamic sampling strategy used in LambdaFM which is a one-hidden layer embedding model has achieved significant success in the recommender system domain. It is natural to adapt the lambda sampling strategy for deep embedding models. However, although we have proposed two ways to perform efficient sampling, the original LambdaFM sampling technique is still very expensive for deep learning models with more hidden layers. The main time complexity comes from the cost of the scoring function (i.e., deep learning models) computation. And thus the major difficulty is how to identify the informative negative examples by bypassing the scoring function computation. Recently, a kernel based sampling method (Blanc and Rendle, 2017) was investigated to approximate the real data distribution without resorting to the computation of the scoring function. However, the method only shows the decreasing trend of loss function without guarantee of recommendation accuracy, such as NDCG and MRR discussed in Chapter 2. We may further exploit the kernel based sampling method and apply it to various deep embedding models such as Neural Factorization Machines (He et al., 2017) and Attentional Factorization Machines (Xiao et al., 2017).

- Batch gradient optimization for various loss functions: The proposed f_{BGD} method has achieved better recommendation accuracy than a variety of negative sampling based methods. However, unlike the negative sampling based methods, f_{BGD} is only limited to the least square regression loss function. So it is interesting to adapt it for other loss functions such as pairwise learning to rank losses (e.g., BPR (Rendle et al., 2009b)). A possible solution is to design a proper surrogate function that is expressed as quadratic function (i.e., $ax^2 + bx + c$). For example, by Taylor series, the exponential loss e^x approximately equals to $1 + x + 1/2x^2$ when x is small (e.g.,

$x < 1.0$). In addition, the proposed f_{BGD} is not limited to the domains discussed in this paper. It potentially benefits many real-world applications with positive-only data, such as genes association studies (Asgari and Mofrad, 2015) and data stream mining (Li et al., 2009).

- Deep learning based methods have achieved huge success in domains such as speech recognition, computer vision and natural language processing. We notice there are many interesting deep neural networks that were originally designed for other domains can potentially benefit the recommender system domain. For example, the recently proposed Caser (Tang and Wang, 2018) which has achieved success in the session-based recommendation task is originally inspired by (Kim, 2014), which is a well-known convolutional neural networks propose for language modeling.

- Generative adversarial network (GAN) has been recently applied into the recommender system domain. The first GAN model, referred to as IRGAN (Wang et al., 2017a), is similar to our idea in LambdaFM as both models rely on the negative sampling strategy. The main difference is that IRGAN consists of two modules (i.e., a generator and discriminator), while LambdaFM has only a discriminator. Interestingly, some work (Wang et al., 2017a; Yu et al., 2017) argues that leveraging both generator and discriminator shows better performance than only using discriminator, while other (Zhao et al., 2017) work empirically shows that using only discriminator performs better. It is interesting to investigate whether the good performance of IRGAN comes from the additional generator.

- Although implicit feedback data is much more popular, explicit feedback may still be available for the top-N item recommendation task. In this thesis, we do not distinguish the preference difference of explicit feedback by simply treating all of them as positive. In practice, there are various strategies to subdivide the observed feedback. For example, an intuitive weighting function can be designed to capture the difference of observed feedback by giving higher values to items with higher ratings or more interaction frequencies. While in this case, our proposed models in this thesis

have to be accordingly extended to satisfy this request. It will be interesting to study whether the recommendation quality can be largely improved by combining the observed explicit feedback.

- In this thesis, we investigate all our proposed recommendation models by the offline protocol. As for future work, we plan to study the performance of them by online protocol with a more realistic recommendation scenario.

8.3 Closing Remarks

Recommender systems have played an important role in improving the problem of information overload. Personalized and context-aware recommendation have attracted much attention from both academia and industry. One of the most important challenges in the recommender system domain is the data sparsity due to the lack of users' explicit/implicit feedback. In this thesis, we focus on item recommendation problem from implicit feedback. We find that recommendation quality and efficiency are largely determined by how to use unobserved feedback. We have introduced two types of approaches for dealing with unobserved feedback, i.e., sampling and non-sampling (or whole-data based) strategies. Generally speaking, sampling based methods are more popular than using whole data since they are generic and not limited to specific models. By contrast, the whole-data based method have shown higher recommendation accuracy than sampling based methods since sampling based methods may have sampling bias and ignore some important examples. An important drawback of the whole-data based method is that they are limited to specific loss and recommendation models. Particularly, they may not be applied to deep learning models with non-linear activation functions. In contrast, the negative sampling strategies have no such limitations. That is, both methods have their own pros and cons. In practice, they should be chosen by taking into account of the specific application.

References

Charu C. Aggarwa. Recommender systems. 2016. https://www.springer.com/gb/book/9783319296579. 4, 11, 12, 13, 14, 16

Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287, 2015. 162

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 147

Ricardo Baeza-Yates, Carlos Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *International Conference on Extending Database Technology*. Springer, 2004. 3

Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1341–1350. International World Wide Web Conferences Steering Committee, 2017. 18, 20, 110

Nicholas J Belkin and W Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12): 29–38, 1992. 2

Betim Berjani and Thorsten Strufe. A recommendation system for spots in location-based online social networks. In *Proceedings of the 4th Workshop on Social Network Systems*, page 4. ACM, 2011. 87

Alberto Bertoni, Paola Campadelli, and M Parodi. A boosting algorithm for regression. In *ICANN*, pages 343–348. Springer, 1997. 67, 68

Guy Blanc and Steffen Rendle. Adaptive sampled softmax with kernel based sampling. *arXiv preprint arXiv:1712.00527*, 2017. 109, 161

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005. 31, 41, 42, 43, 55

Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. 38

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007. 42

Pablo Castells, Saúl Vargas, and Jun Wang. Novelty and diversity metrics for recommender systems: choice, discovery and relevance. 2011. http://ir.ii.uam.es/rim3/publications/ddr11.pdf. 13

Sotirios P. Chatzis, Panayiotis Christodoulou, and Andreas S. Andreou. Recurrent latent variable networks for session-based recommendation. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, pages 38–45. ACM, 2017. 136

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 138

Long Chen, Fajie Yuan, Joemon M Jose, and Weinan Zhang. Improving negative sampling for word representation using self-embedded features. *arXiv preprint arXiv:1710.09805*, 2017. 21

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016. 16

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research*, 13(Dec):3619–3622, 2012. vii, 24, 25, 41, 108, 109, 111, 123, 124, 141

Chen Cheng, Haiqin Yang, Irwin King, and Michael R. Lyu. Fused matrix factorization with geographical and social influence in location-based social networks. In *Association for the Advancement of Artificial Intelligence*, 2012. 85, 86, 87, 88, 89, 90, 93, 99

Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. Where you like to go next: Successive point-of-interest recommendation. In *International Joint Conference on Artificial Intelligence*, 2013. 141

Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 265–272. ACM, 2014. 69

Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan Kankanhalli, and Liqiang Nie. Exploiting music play sequence for music recommendation. In *International Joint Conference on Artificial Intelligence*, 2017. 136

Nipa Chowdhury, Xiongcai Cai, and Cheng Luo. Boostmf: boosted matrix factorisation for collaborative ranking. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–18. Springer, 2015. 67, 68, 70

Konstantina Christakopoulou and Arindam Banerjee. Collaborative ranking with a push at the top. In *Proceedings of the 26th International Conference on World Wide Web*, pages 205–215, 2015. 76, 124

Alberto Costa and Fabio Roda. Recommender systems by means of information retrieval. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 57. ACM, 2011. 2

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010. 41

Qiang Cui, Shu Wu, Yan Huang, and Liang Wang. A hierarchical contextual attention-based gru network for sequential recommendation. *arXiv preprint arXiv:1711.05114*, 2017. 141

Luis M De Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *International Journal of Approximate Reasoning*, 51(7):785–799, 2010. 17

Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. Dynamic matrix factorization with priors on unknown values. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 189–198. ACM, 2015. 119

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 122

Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3):293–318, 2001. 67

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 67, 68

Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003. 41, 42, 55, 67

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, pages 337–407, 2000. 68

Huiji Gao, Jiliang Tang, Xia Hu, and Huan Liu. Exploring temporal effects for location recommendation on location-based social networks. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 93–100. ACM, 2013. 77, 85, 87

Huiji Gao, Jiliang Tang, Xia Hu, and Huan Liu. Content-aware point of interest recommendation on location-based social networks. In *Proceedings of the twenty-ninth conference on uncertainty in artificial intelligence*, 2015. 85

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017. 136

Anupriya Gogna and Angshul Majumdar. A comprehensive recommender system model: Improving accuracy for both warm and cold start users. *IEEE Access*, 3:2803–2813, 2015. 23

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 143

Miha Grčar, Dunja Mladenič, Blaž Fortuna, and Marko Grobelnik. Data sparsity issues in the collaborative filtering framework. In *International Workshop on Knowledge Discovery on the Web*, pages 58–76. Springer, 2005. 6

Youyang Gu, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Learning to refine text based recommendations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2103–2108, 2016. 141, 150

Guibing Guo, Shichang Ouyang, and Fajie Yuan. Approximating word ranking and negative sampling for word embedding. *International Joint Conferences on Artificial Intelligence*, 2018a. 65

Guibing Guo, Songlin Zhai, Fajie Yuan, Yuan Liu, and Xingwei Wang. Vse-ens: Visual-semantic embeddings with efficient negative sampling. *arXiv preprint arXiv:1801.01632*, 2018b. 65

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a. 145, 147

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b. 145, 147

Xiangnan He and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 355–364, 2017. vii, 28

Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558. ACM, 2016c. 18, 20, 110, 114, 153

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on Wold Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017. vii, 16, 27, 28, 29, 32, 108, 125, 161

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Support vector learning for ordinal regression. 1999. 16, 42, 55

Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. *arXiv preprint arXiv:1706.03847*, 2017. 141, 153

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015. 110, 119, 136, 141, 150, 153

Liangjie Hong, Aziz S Doumith, and Brian D Davison. Co-factorization machines: modeling user interests and predicting individual decisions in twitter. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 557–566. ACM, 2013. 55, 108, 109, 110, 113

Longke Hu, Aixin Sun, and Yong Liu. Your neighbors affect your ratings: on geographical neighborhood influence to rating prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 345–354. ACM, 2014. 85, 88, 89, 99

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008. 18, 19, 108, 118, 119

Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation–analysis and evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4):14, 2011. 14, 34

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 147

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014. 149

Xiaotian Jiang, Zhendong Niu, Jiamin Guo, Ghulam Mustafa, Zihan Lin, Baomi Chen, and Qian Zhou. Novel boosting frameworks to improve the performance of collaborative filtering. In *Asian Conference on Machine Learning*, pages 87–99, 2013. 67, 68, 69, 70

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 43–50, 2016. 124

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016. 136, 138, 147

Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010. 18, 41

Rahul Katarya and Om Prakash Verma. An effective collaborative movie recommender system with cuckoo search. *Egyptian Informatics Journal*, 2017. 3

Noriaki Kawamae. Serendipitous recommendations via innovators. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 218–225. ACM, 2010. 3

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 162

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 152

Yehuda Koren. The bellkor solution to the netflix grand prize. 2009. `https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`. 12

Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010. 25

Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015. vii, 3, 24

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009. vi, 15, 16, 18, 20, 22, 23, 109, 116

Mohit Kothari and Wiraatmadja. Reviews and neighbors influence on performance of business. pages 1–10. 103

Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 173–182. ACM, 2012. 88

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011. 142

Phong Le and Willem Zuidema. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive lstms. *arXiv preprint arXiv:1603.00423*, 2016. 120

Mark Levy. Offline evaluation of recommender systems: all pain and no gain? In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, pages 1–1. ACM, 2013. 33

Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1419–1428. ACM, 2017. 141, 143, 150, 151

Longfei Li, Peilin Zhao, Jun Zhou, and Xiaolong Li. A boosting framework of factorization machine. *arXiv preprint arXiv:1804.06027*, 2018. 75

Xiao-Li Li, Philip S Yu, Bing Liu, and See-Kiong Ng. Positive unlabeled learning for data stream classification. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 259–270. SIAM, 2009. 162

Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *EAAI*, pages 785–795, 2008. 68

Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. Rank-geofm: A ranking based geographical factorization method for point of interest recommendation. In *Proceedings of the 37th international*

ACM SIGIR conference on Research & development in information retrieval, pages 433–442, 2015a. 87, 101

Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. Rank-geofm: a ranking based geographical factorization method for point of interest recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 433–442, 2015b. 34, 69, 72, 109, 110

Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 831–840. ACM, 2014. 86, 87, 92, 93

Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003. 15

Bin Liu, Hui Xiong, Spiros Papadimitriou, Yanjie Fu, and Zijun Yao. A general geographical probabilistic factor model for point of interest recommendation. *Knowledge and Data Engineering, IEEE Transactions on*, 27(5):1167–1179, 2015. 88

Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009. 20, 41

Xin Liu, Yong Liu, Karl Aberer, and Chunyan Miao. Personalized point-of-interest recommendation by mining users' preference transition. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 733–738. ACM, 2013. 87, 101

Yong Liu, Peilin Zhao, Aixin Sun, and Chunyan Miao. A boosting algorithm for item recommendation with implicit feedback. 67, 70, 108, 110, 125

Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011. 16

Qiuxia Lu, Tianqi Chen, Weinan Zhang, Diyi Yang, and Yong Yu. Serendipitous personalized ranking for top-n recommendation. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, volume 1, pages 258–265. IEEE, 2012. 34, 49

Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 775–782, 2010. 32, 34, 39, 57

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 30, 31, 108, 109, 110, 113, 116, 119

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 147

Wei Niu, James Caverlee, and Haokai Lu. Neural personalized ranking for image recommendation. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 423–431. ACM, 2018. 20

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a. 136, 138

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b. 136, 142

Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008. 18, 19, 30, 108, 110, 113

Weike Pan and Li Chen. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. 2013. 18, 20, 21, 39, 57, 59, 69, 86

Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 11–18. ACM, 2008. 49

Barak Pearlmutter. Gradient descent: Second order momentum and saturating error. In *Advances in neural information processing system*, pages 887–894, 1992. 109

Wenjie Pei, Jie Yang, Zhu Sun, Jie Zhang, Alessandro Bozzon, and David M.J. Tax. Interacting attention-gated recurrent networks for recommendation. *arXiv preprint arXiv:1711.05114*, 2017. 141

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing*, pages 1532–1543, 2014. 23

István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 71–78, 2010. 119

Runwei Qiang, Feng Liang, and Jianwu Yang. Exploiting ranking factorization machines for microblog retrieval. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1783–1788. ACM, 2013. 38, 39, 43, 58, 66, 123, 124

Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. *arXiv preprint arXiv:1706.04148*, 2017. 136, 141, 143, 150

C Quoc and Viet Le. Learning to rank with nonsmooth cost functions. 19: 193–200, 2007. 40, 42, 44, 45

Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010. vii, 20, 25, 99

Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012. 3, 16, 39, 41, 44, 58, 59, 99, 108, 109, 112, 123, 124, 141

Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 273–282. ACM, 2014. 4, 18, 20, 21, 41, 49, 62, 65, 69, 88, 101, 108, 115

Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 81–90, 2010. 20, 25, 33, 77, 110, 116

Steffen Rendle, Leandro Balby Marinho, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009a. 26, 27, 35, 88

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009b. 4, 5, 18, 20, 31, 32, 33, 40, 41, 44, 55, 57, 58, 69, 86, 91, 92, 93, 94, 96, 104, 109, 110, 113, 124, 125, 161

Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010. 141

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*. 2011. 3

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 108

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. 15

Tom Sercu and Vaibhava Goel. Dense prediction on sequences with time-dilated convolutions for speech recognition. *arXiv preprint arXiv:1611.09288*, 2016. 138

Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 255–262. ACM, 2016. 29

Swapneel Sheth, Nipun Arora, Christian Murphy, and Gail Kaiser. wehelp: a reference architecture for social recommender systems. In *IEEE/ACM International Conference on Automated Software Engineering workshops. IEEE/ACM International Conference on Automated Software Engineering*. NIH Public Access, 2010. 3

Leily Sheugh and Sasan H Alizadeh. A note on pearson correlation coefficient as a metric of similarity in recommender system. In *AI & Robotics (IRANOPEN), 2015*, pages 1–6. IEEE, 2015. 15

Yue Shi. Ranking and context-awareness in recommender systems. 2013. http://homepage.tudelft.nl/c7c8y/theses/PhDThesisShi.pdf. 2, 3

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. TFMAP: optimizing map for top-n context-aware recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 155–164, 2012a. 22, 41, 42

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012b. 22, 32, 34, 35, 39, 41, 57, 88, 101

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. Cars2: Learning context-aware representations for context-aware recommendations. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 291–300. ACM, 2014. 22, 41, 42, 69

Elena Smirnova and Flavian Vasile. Contextual sequence modeling for recommendation with recurrent neural networks. *arXiv preprint arXiv:1706.07684*, 2017. 141, 150

Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 720–727, 2003. 19

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate sequence labeling with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017. 138

Yong Kiam Tan, Xinxing Xu, and Yong Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM, 2016. 143, 150, 151

Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *ACM International Conference on Web Search and Data Mining*, 2018. 135, 137, 143, 150, 153, 162

Nava Tintarev, Matt Dennis, and Judith Masthoff. Adapting recommendation diversity to openness to experience: A study of human behaviour. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 190–202. Springer, 2013. 3

Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970. 89

Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual inter-*

national ACM SIGIR conference on Research and development in information retrieval, pages 383–390. ACM, 2007. 55

Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, Proceedings of the sixth ACM conference on Recommender systems '17. ACM, 2017. 141, 143, 150

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. vii, 26

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 136, 142

Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 109–116. ACM, 2011. 14

Maksims Volkovs and Guang Wei Yu. Effective latent models for binary feedback in recommender systems. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 313–322, 2015. 119

Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhan, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, 2017a. 110, 113, 123, 125, 162

Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 515–524. ACM, 2017b. 159

Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 403–412. ACM, 2015. 141

Yanghao Wang, Hailong Sun, and Richong Zhang. Adamf: Adaptive boosting matrix factorization for recommender system. In *International Conference on Web-Age Information Management*, pages 43–54. Springer, 2014. 67, 68, 69, 70

Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *International Joint Conference on Artificial Intelligence*, pages 2764–2770, 2011. 23, 52, 65

Jason Weston, Chong Wang, Ron Weiss, and Adam Berenzweig. Latent collaborative retrieval. In *arXiv preprint arXiv:1206.4603*, pages 9–16, 2012. 110, 124

Jason Weston, Hector Yee, and Ron J Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 245–248, 2013. 110

Joost Wit. Evaluating recommender systems: an evaluation framework to predict user satisfaction for recommender systems in an electronic programme guide context. Master's thesis, University of Twente, 2008. https://essay.utwente.nl/59711/1/MA_thesis_J_de_Wit.pdf. 14, 16

Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*, 2017. 161

Xin Xin, Yuan Fajie, He Xiangnan, and Jose Joemon. Batch is not heavy: Learning word embeddings from all samples. *The Annual Meeting of the Association for Computational Linguistics*, 2018. 8, 107, 132

Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factoriza-

tion. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010. 20, 25

Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 391–398, 2007. 67, 68, 69, 70, 73

Mao Ye, Peifeng Yin, and Wang-Chien Lee. Location recommendation for location-based social networks. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pages 458–461. ACM, 2010. 90

Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 325–334, 2011. 85, 86, 87, 89

Mao Ye, Xingjie Liu, and Wang-Chien Lee. Exploring social influence for recommendation: a generative model approach. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 671–680, 2012. 85

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 138, 144

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. 2017. 162

Fajie Yuan, Guibing Guo, Joemon Jose, Long Chen, Haitao Yu, and Weinan Zhang. Optimizing factorization machines for top-n context-aware recommendations. In *International Conference on Web Information Systems Engineering*, 2016a. 3, 17, 124

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Lambdafm: learning optimal ranking with factorization machines using

lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 227–236. ACM, 2016b. 4, 5, 8, 17, 18, 20, 21, 33, 39, 49, 66, 98, 108, 109, 110, 113, 119, 123, 124, 125, 126, 128, 158

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Optimizing factorization machines for top-n context-aware recommendations. In *International Conference on Web Information Systems Engineering*, pages 278–293. Springer, 2016c. 158

Fajie Yuan, Joemon M Jose, Guibing Guo, Long Chen, Haitao Yu, and Rami S Alkhawaldeh. Joint geo-spatial preference and pairwise ranking for point-of-interest recommendation. In *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on*, pages 46–53. IEEE, 2016d. 3, 4, 8, 17, 18, 85, 159

Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Boostfm: Boosted factorization machines for top-n feature-based recommendation. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 45–54. ACM, 2017. 4, 8, 17, 18, 66, 98, 108, 110, 113, 119, 123, 125, 126, 158

Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple but hard-to-beat baseline for session-based recommendations. *arXiv preprint arXiv:1808.05163*, 2018a. 3, 9, 160

Fajie Yuan, Xin Xin, Xiangnan He, Guibing Guo, Weinan Zhang, Chua Tat-Seng, and Joemon M Jose. fbgd: Learning embeddings from positive unlabeled data with bgd. *Association for Uncertainty in Artificial Intelligence*, 2018b. 4, 8, 17, 32, 65, 107, 160

Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. Time-aware point-of-interest recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 363–372. ACM, 2013. 85, 86, 87, 90

Jia-Dong Zhang and Chi-Yin Chow. Geosoca: Exploiting geographical, social and categorical correlations for point-of-interest recommendations. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 443–452, 2015. 85, 86, 87

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004. 55, 56

Weinan Zhang. *Optimal real-time bidding for display advertising*. PhD thesis, UCL (University College London), 2016. http://discovery.ucl.ac.uk/1496878/. 13

Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 785–788. ACM, 2013. 47, 108, 110, 113

Shenglin Zhao, Irwin King, and Michael R Lyu. A survey of point-of-interest recommendation in location-based social networks. *arXiv preprint arXiv:1607.00647*, 2016a. 87

Shenglin Zhao, Irwin King, and Michael R Lyu. A survey of point-of-interest recommendation in location-based social networks. *arXiv preprint arXiv:1607.00647*, 2016b. 101

Tong Zhao, Julian McAuley, and Irwin King. Leveraging social connections to improve personalized ranking for collaborative filtering. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 261–270. ACM, 2014. 18, 20, 21, 69, 88, 91, 109, 110

Wei Zhao, Wenyou Wang, Jianbo Ye, Yongqiang Gao, Min Yang, Zhou Zhao, and Xiaojun Chen. Leveraging long and short-term information in content-aware movie recommendation. *arXiv preprint arXiv:1712.09059*, 2017. 159, 162