Implicit Feedback Embeddings for Recommender Systems on Sparse Data

NGUYEN THAI BINH

Doctor of Philosophy

Department of Informatics
School of Multidisciplinary Sciences
The Graduate University for Advanced Studies,
SOKENDAI

Implicit Feedback Embeddings for Recommender Systems on Sparse Data

by

NGUYEN THAI BINH

A dissertation submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of
Doctor of Philosopy

SOKENDAI (The Graduate University for Advanced Studies)
National Institute of Informatics
March 2019

Committee

Advisor:	Dr. Atsuhiro TAKASU Professor of National Institute of Informatics/SOKENDAI
Subadvisor:	Dr. Kenro AIHARA Associate Professor of National Institute of Informatics/SOKENDAI
Examiner:	Dr. Akiko AIZAWA Professor of National Institute of Informatics/SOKENDAI
Examiner:	Dr. Seiji YAMADA Professor of National Institute of Informatics/SOKENDAI
Examiner:	Dr. Yusuke MIYAO Professor of the University of Tokyo

Abstract

In information recommendation, the preferences of users and the attributes of items to be recommended are represented by feature vectors. The items are recommended based on the similarity of the corresponding feature vectors. Characteristics of users and items are learned from the rating for items of users, etc. In general, the amount of ratings is limited, and extracting effective features from sparse data will be necessary. Also, for users who newly participate in the system, it is difficult to obtain user characteristics because the information is limited. The aim of this thesis is to propose effective models of recommender systems for such sparse data.

We first look into the rating prediction problem, one of the essential tasks in recommender systems. Rating is another kind of feedback known as explicit feedback. Different from the implicit feedback, the amount of rating data is limited because it requires users to provide the ratings explicitly. I first proposed a feature extraction method that utilizes the data that is easy to obtain, such as the click history recorded in the log when a user examines an item. In this method, the features of items are extracted based on two sources of feedback: rating data and click data. We show that exploiting the click data can supplement the shortage of rating data.

We further advance this research and proposed a method to analyze the similarity of items in more detail. The proposed model can identify two kinds of relationships between items, (1) “items that can be replaced" such as products of company A and products of company B, and (2) items that are “often purchased together” such as bread and butter. In this research, the item features are extracted under the assumption that there is a strong correlation between the items that are positively evaluated by each user.

Finally, we introduce a model to learn the representations of the products based on their titles to model a collection of shopping transactions. The learned representations can help identify two kinds of relationships between products: the *similar prod-*

ucts and *also-buy products*. The proposed model can be used for multiple purposes: next product recommendation, similar product recommendation, also-buy product recommendation, and product search by keywords. Because the model learns the representations from product titles, it can deal with the cold-start problem, the problem of modeling new products which have not appeared in any shopping transactions.

Contents

List of Figures

List of Tables

List of Abbreviations

LDA	Latent Dirichlet Allocation
MAP	Mean Average Precision
MAR	Mean Average Rank
MRR	Mean Reciprocal Rank
NDCG	Normalized Discounted Cumulative Gain
NMF	Non-negative Matrix Factorization
PMF	Probabilistic Matrix Factorization
RMSE	Root Mean Squared Error
WRMF	Weighted Regularization Matrix Factorization

Acknowledgments

First and foremost, I would like to express my thankfulness to my supervisor Prof. Atsuhiro Takasu for being an incredible advisor and encouraging me during the Ph.D. I also would like to thank Prof. Kenro Aihara, my second supervisor, and my committee members for their valuable feedback, support, and guidance.

I would like to thank the National Institute of Informatics for providing me such a good research environment. I also thank the Japanese government for providing me the scholarship for the Ph.D. course. Many thanks to the members in the Takasu-lab for many interesting and inspiring discussion.

Finally, I would like to thank my parents and my family for their continuing support. Many thanks go to my wife and my kids. They are the motivations for me to finish this Ph.D.

Chapter 1

Introduction

1.1 Motivation

In recent years, recommender systems (RS) have become a core component of many online services. Given a specific user, the goal of an RS is to provide a personalized recommendation of products that he or she may have interest. Common examples of applications include the recommendation of movies (Netflix, Amazon Prime Video), music (Pandora), videos (YouTube), news content (Outbrain) or advertisements (Google).

Understanding users and products is crucial in making good recommendations. Typically, recommender systems learn user and item representations from the user-item interaction data (e.g., rating data or shopping cart data). For example, one of the essential problems in recommender systems is the rating prediction task. Given the historical ratings of users to items, the task is to predict the unseen rating of a given user to a given item. By predicting the rating of a user to an item, a recommender system can understand the preferences of the user to the items and can suggest the items that he or she may like. This requires understanding both the preferences of users as well as items from the historical interactions between users to items.

As another example, we consider the recommendation for online shoppers. We want to recommend the products that a shopper would insert to his/her current shopping cart. We recommend not only the product already in the historical collection of the shopping carts but also want to recommend the new products which have not appeared in any previous shopping carts. This requires understanding both the content of the products from the texts (e.g., product titles and descriptions), as well as the

relationships between products that appear together in the shopping carts.

In this thesis, we aim to learn effective representations of users and items from the real world data (e.g., movie data, music data, online retail data) by applying the tools of probabilistic models, or more precisely, latent variable models, and deep learning techniques. Probabilistic models help reason about the uncertainty of the preferences of users to items, while deep learning provides the flexibility to model complex interactions in the data.

1.2 Contributions of This Work

Typically, the user-item interaction data is extremely sparse. In the rating data, the density of the rating matrix is often less than 0.1%. In the purchase data, each user buys only a handful number of products among a huge amount of the products. Particularly, only one or two products are observed in the purchase history of many users. Due to the sparsity of the user-item interaction data, the traditional models such as matrix factorization or user-based collaborative filtering model are less accurate. There are some research that try to address this problem by exploiting auxiliary information such as textual data (Wang and Blei, 2011; Wang et al., 2015a), visual data (He and McAuley, 2016), or audio signals (Oord et al., 2013). However, these content models still rely on the observed data, and if the observed data (the interaction data) is extremely sparse, a good content model cannot be obtained. Furthermore, in many cases, the auxiliary data is not easy to collect. In this research, we focus on studying the embedding models, the models for learning the useful representations of users and items from the sparse data. These embedding models are used in essential tasks of the recommender systems: rating prediction, top-n recommendation, and session-based recommendation for online shoppers.

In details, we propose three embedding models as follows.

1. We propose a probabilistic embedding model for the click data. The proposed embedding model extracts the relationships between the items that are often clicked together by the same users. We then propose CoMF, a joint model of the propose embedding model and the probabilistic matrix factorization for the rating prediction task. By this way, the model can learn the item representations reflecting the item relationships and are useful for rating prediction. This can help address the sparsity of the rating matrix. The proposed model shows

a better result than competing methods, in addition to the favorite performance in the cold-start case. The high-level architecture of the proposed model and the CoMF is shown in Fig.1.1.

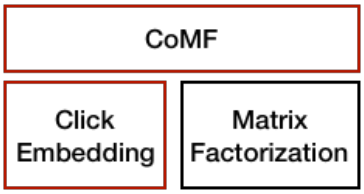


Figure 1.1: The high-level architecture of the proposed click embedding and the joint model CoMF. The black lined block is the existing model, the red lined blocks are the proposed models. Here the click embedding and the matrix factorization are the building block of the CoMF.

2. We propose a neural personalized embedding model which is effective in making recommendations for cold-users, i.e., the users who have few historical data. In recommending a product to a user, we consider two signals: the preference of the user to the product, and the compatibility between this product and other products that the users already consumed. When the user is not well understood, the relationships between the candidate product and other products that he/she already consumed will help provide a good suggestion. The high-level architecture of the proposed NPE model is shown in Fig.1.2.

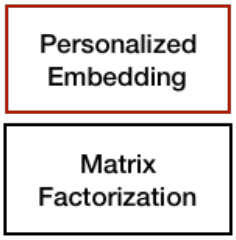


Figure 1.2: The high-level architecture of the proposed personalized neural embedding model (NPE). The black lined block is the existing model, the red lined block is the proposed model. Here the NPE is built upon the matrix factorization by introducing the co-occurrence information of the items.

3. We propose a model to learn the representations of the products based on their titles to model a collection of shopping transactions. First, we propose a sequential embedding model that capture the relationships between the products that are often added to the same baskets. Then we propose BASTEXT, which is a combination of this sequential model and the textual content model (e.g.,

convolutional neural network, LSTM) for learning the representations of the products from shopping basket data and product titles. The learned representations can help identify two kinds of relationships between products: the *similar products* and *also-buy products*. The proposed model can be used for multiple purposes: next product recommendation, similar product recommendation, also-buy product recommendation, and product search by keywords. Because the model learns the representations from product titles, it can deal with the cold-start problem, the problem of modeling new products which have not appeared in any shopping transactions. The high-level architecture of the BASTEXT is shown in Fig.1.3.

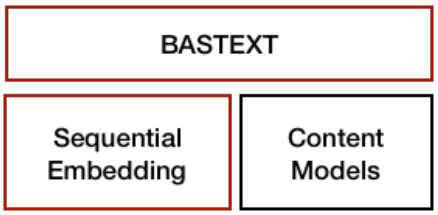


Figure 1.3: The high-level architecture of the proposed BASTEXT model. The black lined block is the existing model, the red lined block are the proposed models. Here the BASTEXT is built upon the content model and the sequential embedding model.

1.3 Organization of This Thesis

Chapter 2

This chapter discusses some background knowledge and previous work that are helpful in understanding the rest of this thesis. Broadly speaking, we will make use of matrix factorization techniques for recommender systems and word embedding techniques for modeling words and sentences.

Chapter 3

We introduce a model that efficiently combines explicit and implicit feedback in a unified model for rating prediction. This model is a combination of matrix factorization and item embedding, a similar concept with word-embedding in natural language processing. We also describe the experiments on three real-datasets (Movie-

lens 1M, Movielens 20M, and Bookcrossing) to demonstrate the advantages of the model in rating prediction, particularly for sparse datasets.

Chapter 4

We introduce a neural personalized embedding (NPE) model, which improves the recommendation performance for cold-users and can learn effective representations of items. It models a user's click to an item in two terms: the personal preference of the user for the item, and the relationships between this item and other items clicked by the user. The evaluation by both quantitative and qualitative experiments shows the advantages of the model.

Chapter 5

We introduce a model to learn the representations of the products of a collection of shopping baskets from their titles. The learned representations can help identify two kinds of relationships between products: the *similar products* and *also-buy products*. The proposed model can be used for multiple purposes: next product recommendation, similar product recommendation, also-buy product recommendation, and product search by keywords.

Chapter 2

Background and Related Work

This chapter discusses some background knowledge and previous work that is helpful for understanding the rest of this thesis. Broadly speaking, we will make use of matrix factorization techniques for recommender systems, word embedding techniques for modeling words and sentences.

2.1 Recommender Systems

Modern recommender systems (RS) are a core component of many online services. An RS analyzes users' behavior and provides them with personalized recommendations for products or services that meet their needs. For example, Amazon recommends products to users based on their shopping histories; an online newspaper recommends articles to users based on what they have read. Broadly speaking, recommender systems use the historical data, i.e., the feedback, to infer users' preferences and use the inferred preferences to suggest items.

In recommender systems, there is a set of users and a set of items (e.g., movies, songs, products, articles) and the feedback from the users (e.g., ratings, like/dislike, clicks, views, purchases). The feedbacks are also referred to as the interactions between users and items (user-item interactions). In addition to the user-item interactions, additional data may be available such as textual contents (e.g., product descriptions), visual contents (e.g., product images) and demography information.

2.1.1 Types of Recommender Systems

Generally, an RS can be classified into two categories: Content-based approach and collaborative filtering-based (CF-based) approach. The content-based approach creates a description for each item and builds a profile for each user's preferences. In other words, the content-based approach (Pazzani and Billsus, 2007) recommends items that are similar to items for which the user has expressed interest in the past. In contrast, the CF-based approach (Das et al., 2007; Koren, 2008; Salakhutdinov and Mnih, 2008) relies on the past behavior of each user, without requiring any information about the items that the users have consumed. The consensus is that the CF-based methods generally outperform the content-based methods if there is enough feedback (e.g., ratings, clicks).

2.1.2 Types of Feedback

The feedbacks of users can be *explicit* (e.g., rating scores/stars, like/dislike) or *implicit* (e.g., click, view, purchase). The explicit feedbacks explicitly reflect the opinion of users to items. For example, a user clicks to like/dislike explicitly express that he/she likes or dislikes a product; or a user rates movies on a scale, e.g., 1-5 stars. The advantage of explicit feedback is that it can express the preference of a user to an item that he/she likes or dislikes the item. This is important to capture the taste of a user. However, one disadvantage of the explicit feedback is that it is difficult to collect, thus usually the explicit data is extremely sparse.

The implicit feedback, in contrast, is collected when users use the systems, for example when a user clicks a product or when a user reads an article. Different from the explicit feedback which is represented by both positive and negative values (e.g., like/dislike), the implicit feedback is represented by positive values. Usually, we use 1 to indicate that interaction actually happened, e.g., a user clicked/purchased a product, or a user viewed an article. The advantage of the implicit feedback data is that it is easy to collect and usually is abundant. However, the disadvantage is that it does not directly express the preference of a user to an item. For example, even if a user purchased an item, it does not mean that the user likes the item. In addition, if there is no interaction between a user to an item, it does not necessarily imply that the user dislikes the item. It may be because the user is not aware of the existence of the item.

2.2 Related Work

2.2.1 Collaborative filtering

Most of the modern recommender systems rely on collaborative filtering (CF), which learns user preferences from their prior behaviors such as ratings, purchases, or clicks. One of the most efficient methods for CF is matrix factorization (MF) which models user preferences based on the user-item interaction matrix (e.g., rating matrix, click matrix) (Hu et al., 2008; Koren, 2008; Pan et al., 2008; Salakhutdinov and Mnih, 2008). Given prior ratings of users to items, MF learns the latent feature vectors of users and items and uses these vectors to predict missing ratings. Recently, MF is improved by replacing the inner products of the latent factor vectors of the users and items with a feed-forward neural network (Dziugaite and Roy, 2015; He et al., 2017).

The rating prediction suffers from the sparseness of the rating matrix. To address sparseness in the user-item matrix, additional data about items/users are also used (Oord et al., 2013; Wang and Blei, 2011; Wang et al., 2015a). Implicit feedback is also exploited to address this problem. The idea of using implicit feedback to boost the performance of rating prediction is first introduced in (Bell and Koren, 2007) and SVD++ (Koren, 2008). In these models, the authors integrate implicit feedback which is in the form "who rates what" into the factorization model. The implicit feedback that these models used is inferred from the rating data, therefore they can not model an item if it does not have any rating. Our approach is different, we use implicit feedback as an independent data source, therefore our model can model an item even if it does not have any rating. Another difference is that the implicit feedback used in (Bell and Koren, 2007) and (Koren, 2008) is inferred from rating data, therefore it is also very sparse. The implicit feedback used in our model is denser and we can exploit more information from the implicit feedback.

Co-rating (Liu et al., 2010) combines explicit and implicit feedback by treating explicit feedback as a special kind of implicit feedback. The explicit feedback is normalized into the range $[0, 1]$ and is summed up to implicit feedback matrix with a fixed proportion to form a single matrix. This matrix is then factorized to obtain the latent vectors of users and items. The main difference between Co-rating and our method is that the contribution of implicit feedback and explicit feedback in the representation of items is equal for every item, while in our approach this proportion

is dynamic. For an item to which there are a lot of ratings available, its representation mainly comes from the explicit feedback. On the other hand, if an item has few or does not have any rating data, its representation mainly comes from the implicit feedback data.

Wang et. al.(Wang et al., 2012) proposed Expectation-Maximization Collaborative Filtering (EMCF) which exploits both implicit and explicit feedback for the recommendation. For predicting ratings for an item to which rating data is not available, the rating is inferred from ratings of its neighbor regarding click data. However, the algorithm is based on an iterative Expectation-Maximization (EM) in which E-phase is a matrix factorization model. In other words, it needs multiple times of matrix factorization and therefore is not efficient in computation.

Collective matrix factorization (CMF) (Singh and Gordon, 2008) proposed a framework for factorizing multiple related matrices simultaneously, in order to exploit information from multiple sources. For example, if the item-genre matrix exists, one can factorize both user-item and item-genre matrices in a shared latent space. This approach is able to incorporate the side information (e.g., genre information of items) into the latent factor model. Our model is a special case of CMF with the rating matrix and item-item co-occurrence matrix.

Recently, the CoFactor (Liang et al., 2016) and CEMF (Nguyen et al., 2017) models have been proposed. These models integrate item embedding into the MF model. They simultaneously decompose the preference matrix and the SPPMI matrix (the item-item matrix constructed from co-click information) in a shared latent space. However, in contrast to our proposed method, CoFactor and CEMF use co-click information to regularize the user-item matrix information, whereas NPE exploits co-click information for learning effective representations of items. In (Nguyen and Takasu, 2017), the author uses co-click information to address the data sparsity issue in rating prediction.

For cold-user recommendations, (Tang and Liu, 2017) and (Li et al., 2015b) proposed models that learn user presentations from user profiles. In (Tang and Liu, 2017), the user representations are learned from user profiles via a deep convolutional neural network for event recommendations, whereas (Li et al., 2015b) has user representations being learned by an auto-encoder. Despite these models being very useful for new-user recommendations, the main issue remains that user profiles are not always available. Furthermore, many user profiles may be very noisy (e.g.,

users may not want to publish their real gender, age, or location), which leads to inaccurate representations of users.

2.2.2 Sequential Recommendation

Another line of recommendation is *sequential recommendation* which considers the interactions of users to items as a sequence with an explicit order, e.g., a sequence of clicks. A common approach to this problem is the Markov chain-based method. An example of this approach is Markov Decision Process (MDP) (Shani et al., 2005) for predicting next action given last actions, Markov Embedding (Chen et al., 2012b) for playlist generation. Recently, recurrent neural network-based approaches are also introduced to this problem (Hidasi et al., 2016). Note that, our problem is different from sequential recommendation. In shopping basket modeling, there is no explicit order in which the products are added to the baskets. Although a customer adds products to his/her basket sequentially, the order in which the products are added to the basket does not change the nature of the basket.

2.2.3 Embedding Models

Word embedding is an approach to learn low-dimensional vector representations of words that capture the relationships between a word with its surrounding words (i.e., the context). These techniques (Li et al., 2015a; Mikolov et al., 2013b) have been applied successfully to many tasks in natural language processing. The goal of word embedding is to learn vector representations of words that capture the relationships with surrounding words. The assumption behind word embedding techniques is that words that occur in the same context are similar. To capture such similarities, words are embedded into a low-dimensional continuous space. Word embedding can be performed by shallow neural network (Mikolov et al., 2013b) or by factorizing a word-word matrix according to their occurrence statistics (Levy and Goldberg, 2014; Pennington et al., 2014). Word embedding is a building block of many sentence or paragraph embedding techniques.

If an item is viewed as a word, and a list of items clicked by a user is a context window, we can map word embedding to recommender systems. *Item2Vec* (Barkan and Koenigstein, 2016) was introduced as a neural network-based item-embedding model. However, Item2Vec is not able to predict missing entries in a user-item matrix

directly. Furthermore, in its recommendations, Item2Vec relies only on the last item, ignoring previous items that a user has clicked.

Exponential Family Embeddings (EFE) (Rudolph et al., 2016), a probabilistic embedding model that generalizes the spirit of word embedding to other kinds of data, which can be used for modeling clicks and learn item representations. However, EFE does not support for side information such as items' rich contents. In addition, EFE is not personalized.

Early approaches on text representation focus on *bag-of-words* (BoW) representation (Manning et al., 2008). Despite that this is a simple common used method, one of its disadvantages is the size of the embedding vectors is proportional to the size of the vocabulary, resulting in high and sparse representations, which is not efficient in computation. Many approaches have been proposed to improve the performance of BoW including low-rank factorization models such as Latent Semantic Indexing (Deerwester et al., 1990), or topic models such as Latent Dirichlet Allocation (Blei et al., 2003). In this approaches, texts are represented by low-dimensional dense vectors, which outperform BoW model in computational complexity as well as in predictive accuracies in natural language processing (NLP) tasks.

Recently, distributed representation learning approaches archive tremendous success in learning text representations. Such approaches range from simply composition of the word vectors (Mikolov et al., 2013b; Pennington et al., 2014) to more complicated network architectures. Approaches in this category include doc2vec (Le and Mikolov, 2014); CNN-based approaches (Kim, 2014; Weston et al., 2014); RNN-base approaches (Tai et al., 2015). Skip-through (Kiros et al., 2015) is another line of text representations, which learn sentence representations by predicting the surrounding sentences of a given one.

2.2.4 Shopping Basket Analysis

By far, the most common approach to shopping basket analysis is *association rules*, which discovers the rules in the form: "*Consumers who buy diapers are likely to buy beer*". Formally, such rules can be expressed as $B => i$, where B is a set of products and i is a product not contained in B. Such rules are useful in making recommendations given the products currently in the basket.

Another direction of basket analysis is *next basket* recommendations (Rendle et

al., 2010; Wang et al., 2015b; Yu et al., 2016) which is to suggest a specific customer a whole basket, given his previously shopping transaction. Our work is different. We are not working on recommending next basket, we are focusing on recommending the next product to add to the current basket.

A relevant technique to the basket-sensitive recommendation is item-to-item recommendation (Linden et al., 2003; Sarwar et al., 2001) in which a product is recommended based on the similarity between it a product in the shopping cart. Technically, an item-item matrix is built from the available shopping baskets. The items that frequently co-occur in shopping baskets are deemed to be similar and their corresponding elements in the item-item matrix should be large. The value of item-item matrix is used for recommending the next item of a particular one. While this is a simple-but-effective approach, it recommends products based only on the last product, ignoring the information from other products in the basket.

Chapter 3

Implicit Feedback Embedding for Rating Prediction

Collaborative filtering (CF) is one of the most efficient ways for recommender systems. Typically, CF-based algorithms analyze users' preferences and items' attributes using one of two types of feedback: *explicit feedback* (e.g., the ratings given to items by users, like/dislike) or *implicit feedback* (e.g., clicks, views, purchases). Explicit feedback is reliable but is extremely sparse; whereas implicit feedback is abundant but is not reliable. To leverage the sparsity of explicit feedback, in this work, we propose a model that efficiently combines explicit and implicit feedback in a unified model for rating prediction. This model is a combination of *matrix factorization* and *item embedding*, a similar concept with word-embedding in natural language processing. The experiments on three real-datasets (*Movilens 1M*, *Movielens 20M*, and *Bookcrossing*) demonstrate that our method can efficiently predict ratings for items even if the rating data is not available for them. The experimental results also show that our method outperforms competing methods on rating prediction task in general as well as for users and items which have few ratings.

3.1 Introduction

Nowadays, recommender systems (RS) have become a core component of many online services. RS analyzes users' behavior and provides them with personalized recommendations for products or services that meet their needs. For example, Amazon recommends products to users based on their shopping histories; an online newspa-

per recommends articles to users based on what they have read.

Generally, an RS can be classified into two categories: Content-based approach and Collaborative Filtering-based (CF-based) approach. The content-based approach creates a description for each item and build a profile for each user's preference. In other words, the content-based approach recommends the items that are similar to items that interested the user. In contrast, CF-based approach (Koren, 2008, 2009; Ning et al., 2015; Salakhutdinov and Mnih, 2008; Sarwar et al., 2001) relies on the behavior of each user in the past, such as users' ratings on items. The CF-based approach is domain-independent and does not require content collection as well as content analysis. In this work, we focus on the CF-based approach.

Basically, the data for a CF-based algorithm comes in the form of a rating matrix whose entries are observed ratings to items given by users. This kind of feedback is referred to as *explicit feedback*. Given these observed ratings, a typical task of CF-based algorithms is to predict the unseen ratings. One of the most efficient ways to perform CF is matrix factorization (MF) (Koren, 2008, 2009; Salakhutdinov and Mnih, 2008) which decompose the rating matrix into latent vectors that represent *users' preferences* and *items' attributes*. These latent vectors are then used to predict the unseen ratings. Usually, MF-based algorithms suffer from the sparseness of the rating matrix: if a user or an item has a very few numbers of ratings, it is difficult to find a "right" latent vector for the user or the item; in an extreme case, if rating data is not available for an item, MF-based algorithms cannot find a latent vector for it.

When the ratings are not sufficient for modeling the items, exploiting other side information is a solution. Collaborative topic model (Wang and Blei, 2011) and content-based Poisson factorization (Gopalan et al., 2014) use text content information of items as a side information for recommending new items. Collaborative deep learning (Wang et al., 2015a) models music content by a deep neural network and combine with an MF-based model for music recommendation. However, in many cases, the items' contents are not available or are not informative enough for modeling the items (e.g., when an item is described by a very short text, or only by some keywords).

In this chapter, instead of using such side information of items' contents, we focus on utilizing another type of feedback, known as *implicit feedback* (e.g., clicks, views or purchase history), which can be easily collected with abundance during the interaction of users to the system. One way for recommending items that have no

ratings is performing MF model based entirely on the implicit feedback only (Hu et al., 2008; Pan et al., 2008; Rendle et al., 2009). However, implicit feedback only is not reliable in capturing users' preferences because it does not directly express the opinions of users to items. For example, a user's click on an item does not mean that he/she likes the item; it may be the case that the user finally finds that he/she does not like it after clicking it. On the other hand, a user did not click an item may not be because he/she does not like the item, it may be because he/she is unaware about the existence of it. Therefore, using implicit feedback only is not reliable for inferring the preferences of users or attributes of items.

To address the challenges above, we propose a probabilistic model that efficiently combines explicit feedback and implicit feedback in a unified model. We aim to make the model capable of modeling an item using both information from explicit feedback and implicit feedback data. For items that have many ratings, the representation of the item is mainly inferred from the rating data because it is more reliable. On the other hand, for items that have few or does not have any ratings, the recommendation is mainly based on its implicit feedback data.

To find the representation for items without rating data, we exploit the implicit feedback with the assumption: if two items are usually clicked in the same context of each other, they are similar. This technique is referred to as *item embedding* which is very similar to word-embedding techniques (Le and Mikolov, 2014; Levy and Goldberg, 2014; Mikolov et al., 2013a) in natural language processing, which represents each by vectors that capture the relationships with its surrounding words.

In detail, our approach is a combination of two components: (i) the *matrix factorization* model (Salakhutdinov and Mnih, 2008) for finding representations of users and items based on explicit feedback, and (ii) the *item embedding* model for finding the representations of items that can capture the relationships among items based on implicit feedback. We will develop a probabilistic model that jointly learns the MF and item embedding model together. The item representations from item embedding model are learned to adapt with the item representations in the MF model, and then can be used for rating prediction.

The rest of this section is organized as follows. In Section 3.2 we formulate the problem and represent the background knowledge related to the method. Section 3.3 represents our idea in modeling items using implicit feedback and describes the probabilistic model for integrating implicit and explicit feedback data in a unified

model. In Section 3.4, we present the effectiveness of our method by comparing with state-of-the-art techniques using three public datasets. We summarize the method as well as discuss some potential future directions in Section 3.5.

3.2 Preliminary

3.2.1 Notation and Problem Formation

Let us establish some notations. We use u to denote a user and i or j to denote an item. Each observation of explicit feedback is represent by a triplet (u, i, r_{ui}) where r_{ui} is the rating that user u gave to item i. The explicit feedback can be represented by a matrix $R \in \mathbb{R}^{N \times M}$ where N is the number of users and M is the number of items. Each entry r_{ui} of the matrix R is either the rating of item i given by user u or zero if the rating is not observed (missing entries). We use $\mathcal{R}$ to denote the set of (u, i)-pair that $r_{ui} > 0$, $\mathcal{R}_u$ to denote the set of item that user u gave ratings, and $\mathcal{R}_i$ to denote the set of users that gave ratings to item i.

The implicit feedback of a user-item pair is represented by a triplet (u, i, p_{ui}) where $p_{ui} = 1$ if implicit feedback of user u to item i is observed (i.e., the click, views or purchase of item i by user u), and $p_{ui} = 0$ if the implicit feedback is not observed. Implicit feedback is represented by matrix $P \in \{0, 1\}^{N \times M}$.

Usually, explicit feedback matrix R is very sparse (i.e., with many missing entries). We are interested in predicting the missing entries of R based on the observed data.

The notations used in this section are summarized in Table 3.1.

3.2.2 Probabilistic Matrix Factorization

Probabilistic matrix factorization (PMF) (Salakhutdinov and Mnih, 2008) is a method for modeling ratings which represents users and items by vectors in a shared latent space: user u is represented by vector $\mathbf{x}_u \in \mathbb{R}^d$ ($u = 1, 2, \ldots, N$) and item i is represented by vector $\mathbf{y}_i \in \mathbb{R}^d$ ($i = 1, 2, \ldots, M$), where d is the dimension of the latent space.

PMF assumes that the rating r_{ui} can be modeled by a normal distribution as follows.

$$P(r_{ui}|\mathbf{x}_u, \mathbf{y}_i, \sigma_r^2) \propto \mathcal{N}(\mathbf{x}_u^\top \mathbf{y}_i, \sigma_r^2) \tag{3.1}$$

Table 3.1: The notations used throughout the Chapter 3.

Notation	Meaning
N, M	the number of users and items, respectively
R	the rating matrix matrix ($R \in \mathbb{R}^{N \times M}$)
$\mathcal{R}_u$	the set of items that user u provided ratings
$\mathcal{R}_i$	the set of users that provided ratings to item i
d	the dimensionality of the latent space
$\mathbf{x}_u$	the latent factor vector of user u ($\mathbf{x}_u \in \mathbb{R}^d$)
$\mathbf{y}_i$	the latent factor vector of item i ($\mathbf{y}_i \in \mathbb{R}^d$)
$\mathbf{w}_i$	the embedding vector of item i ($\mathbf{w}_i \in \mathbb{R}^d$)
$\mathbf{z}_i$	the embedding vector of item i ($\mathbf{z}_i \in \mathbb{R}^d$)
b_u	the bias term of user u ($b_u \in \mathbb{R}$)
c_i	the bias term of item i ($c_i \in \mathbb{R}$)
μ	the global mean of the rating data ($\mu \in \mathbb{R}$)
$\mathbf{\Theta}$	The set of all model parameters

where θ_u and $\mathbf{y}_i$ are random variables that are drawn from multivariate Gaussian distributions:

$$\begin{aligned} \mathbf{x}_u &\propto \mathcal{N}(0, \sigma_X^2 \mathbf{I}_d) \\ \mathbf{y}_i &\propto \mathcal{N}(0, \sigma_Y^2 \mathbf{I}_d) \end{aligned} \tag{3.2}$$

The graphical model of the PMF is shown in Fig.3.1

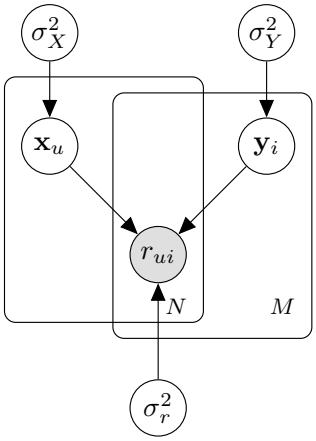


Figure 3.1: The graphical model of the PMF.

The parameters of the model ($\mathbf{x}_{1:N}, \mathbf{y}_{1:M}$) are learned by maximizing the log

posterior distribution which is given in Eq.(3.3).

$$\log P(\mathbf{x}_{1:N}, \mathbf{y}_{1:M}|R, \boldsymbol{\sigma}^2) = -\frac{1}{2\sigma_R^2}\sum_{(u,i)\in\mathcal{R}}(r_{ui}-\mathbf{x}_u^\top\mathbf{y}_j)^2-\frac{1}{2\sigma_X^2}\sum_{u=1}^{N}||\mathbf{x}_u||^2-\frac{1}{2\sigma_Y^2}\sum_{i=1}^{M}||\mathbf{y}_i||^2+C \tag{3.3}$$

where $\boldsymbol{\sigma} = (\sigma_R^2, \sigma_X^2, \sigma_Y^2)$, and C is a constant when $\sigma_R, \sigma_X, \sigma_Y$, are fixed.

Maximizing Eq.(3.3) is equivalent to minimizing the following error function.

$$\mathcal{L}(\mathbf{x}_{1:N}, \mathbf{y}_{1:M}) = \frac{1}{2}\sum_{(u,i)\in\mathcal{R}}(r_{ui} - \mathbf{x}_u^\top\mathbf{y}_i)^2 + \frac{\lambda_X}{2}\sum_{u=1}^{N}||\mathbf{x}_u||_F^2 + \frac{\lambda_Y}{2}\sum_{i=1}^{M}||\mathbf{y}_i||_F^2 \tag{3.4}$$

where $||.||_F$ is the Frobenius norm of a vector, $\lambda_X = \sigma_R^2/\sigma_X^2$, $\lambda_Y = \sigma_R^2/\sigma_Y^2$. We can see that maximizing the log posterior is equivalent to minimizing the squared loss between the predicted ratings and the actual ratings with L_2 regularization.

Notice that when either the user feature vectors or item feature vectors are fixed, the loss function becomes quadratic so its global minimum can be easily computed. This suggests using the alternating least square (ALS) for solving the problem. The idea behind the ALS method to update one of the latent factor vectors $\mathbf{x}_u$ or $\mathbf{y}_i$ while the remaining parameters are fixed. In updating $\mathbf{x}_u$, we take the partial derivative with respect to $\mathbf{x}_u$ while the remaining parameters are fixed, and set this partial derivative to zero to obtain the new value of $\mathbf{x}_u$. Updating $\mathbf{y}_i$ is performed in a similar way. The update rules of $\mathbf{x}_u$ and $\mathbf{y}_i$ are given in the following equations.

$$\begin{aligned}\mathbf{x}_u^{new} &= \Big(\sum_{i\in\mathcal{R}_u}\mathbf{y}_i\mathbf{y}_i^\top + \lambda_X\mathbf{I}_d\Big)^{-1}\Big(\sum_{i\in\mathcal{R}_u}r_{ui}\mathbf{y}_i\Big)\\ \mathbf{y}_i^{new} &= \Big(\sum_{u\in\mathcal{R}_i}\mathbf{x}_u\mathbf{x}_u^\top + \lambda_Y\mathbf{I}_d\Big)^{-1}\Big(\sum_{u\in\mathcal{R}_i}r_{ui}\mathbf{x}_u\Big)\end{aligned} \tag{3.5}$$

The method can be summarized as in the Algorithm 1. As we can see from the above equations, the new value of $\mathbf{x}_u$ depends only on the latent factor vectors of the items, and is independent from the latent factor vectors of other users. This enables to update the $\mathbf{x}_u$ ($u \in \{1, 2, \ldots, N\}$) parallelly. Similarly, we also can paralellize the updates of the item latent factor vectors $\mathbf{y}_i$ ($i \in \{1, 2, \ldots, M\}$).

Algorithm 1: The ALS algorithm for the probabilistic matrix factorization.

Input : Rating matrix: $\mathbf{R}$, regularization parameters: λ_X, λ_Y
Output: The latent factor vectors $\mathbf{x}_{1:N}, \mathbf{y}_{1:M}$

Randomly initialize $\mathbf{x}_{1:N}, \mathbf{y}_{1:M}$ from Gaussian distributions
for *epoch=1 ... T* **do**
    for *u=1 ... N* **do**
        Update user latent factor vector $\mathbf{x}_u$ by Eq.(3.5)
    end
    for *i=1 ... M* **do**
        Update item latent factor vector $\mathbf{y}_i$ by Eq.(3.5)
    end
end

3.3 Proposed method

In this section, we describe our method, which is a combination of probabilistic matrix factorization for rating prediction and item embedding model for implicit feedback data. First, we will present the idea of item embedding model.

3.3.1 Item embedding model based on implicit feedback

Inspired by word embedding techniques (Le and Mikolov, 2014; Levy and Goldberg, 2014; Li et al., 2015a; Mikolov et al., 2013a,b) which represent a word by vectors that capture the relationship with its surrounding words, we apply the same idea to find representations of items based on implicit feedback data.

Similar to words, items also have their *contexts*, which is a model choice and can be defined in different ways. For example, the context can be defined as the set of items that are clicked by a user (user-based context); or can be defined as the items that are clicked in a session with a given item (session-based context). In this work, we use the user-based context, which is defined as follow: given an item i that is clicked by user u, the context of i is the list of all items that u have clicked.

The item embedding model presented in this section is partly based on the word-embedding model presented in (Li et al., 2015a) which we bring into the world of items with a change: in (Li et al., 2015a), each word is represented by a unique vector while in this item embedding model we use two vectors to represent each item. We found that a model that uses two vectors for representing items can be efficiently trained by parallelizing the algorithm for the optimization problem (see

our discussion on parameter learning in Section 3.3.2).

Each item is represented by two vectors: an *embedding vector* $\mathbf{w}_i$ and a *context vector* $\mathbf{z}_i$. These two vectors have different roles: the embedding vector governs the distribution of the item, while the context vector governs the distribution of its context items.

The model describes the appearance of an item conditional on other items in its context as follows.

$$p(i|j) = f(i,j)p(i) \tag{3.6}$$

where $p(i)$ is the probability that item i appears in the data, $p(i|j)$ is the probability that i appear in context of j and $f(i,j)$ is the link function that reflects the association between i and j. The role of the link function is straightforward: if item i is often clicked (i.e., $p(i)$ is high), however, i and j are not often clicked together (i.e., $p(i|j)$ is low) then the link function $f(i,j)$ should have small value. On the other hand, if i is rarely clicked (i.e., $p(i)$ is low) but if i and j are often clicked together (i.e., $p(i|j)$ is high), the link function $f(i,j)$ should have high value.

There are different choices for the link functions which lead to different embedding models. Here, we choose the link function $f(i,j) = \exp\{\mathbf{w}_i^\top \mathbf{z}_j\}$. Combining with Eq.(3.6) we have: $p(i,j) = \exp\{\mathbf{w}_i^\top \mathbf{z}_j\}$, or:

$$\log \frac{p(i|j)}{p(i)} = \mathbf{w}_i^\top \mathbf{z}_j \tag{3.7}$$

Note that $\log \frac{p(i|j)}{p(i)} = \log \frac{p(i,j)}{p(i)p(j)}$ is the point-wise mutual information (PMI) (Church and Hanks, 1990) of i and j, Eq.(3.7) can be rewritten as follows.

$$\mathbf{w}_i^\top \mathbf{z}_j = PMI(i,j) \tag{3.8}$$

Empirically, PMI can be estimated using the actual number of observations in the implicit feedback data.

$$PMI(i,j) = \log \frac{\#(i,j)|\mathcal{D}|}{\#(i)\#(j)} \tag{3.9}$$

where $\mathcal{D}$ is the set of all *item-context* pairs that are observed in the click history of any user, $\#(i)$ is the number of users who clicked item i, $\#(j)$ is the number of users who clicked j, and $\#(i,j)$ is the number users who clicks both i and j.

From Eq.(3.7) and Eq.(3.9) we can observe that, the item vectors and context vectors can be obtained by factorizing the matrix whose elements are defined in Eq.(3.9).

A practical issue arises here: for item pair (i, j) that are less often clicked by the same user, $PMI(i, j)$ is negative, or if they have never been clicked by the same user, $\#(i, j) = 0$ and $PMI(i, j) = -\infty$. However, a negative value of PMI does not necessarily imply that the items are not related. The reason may be because the number of items is very huge, and a user who clicks i may not know about the existence of j. A common way in natural language processing is to replace the negative values by zeros to form the positive PMI (PPMI) matrix (Bullinaria and Levy, 2007). The PPMI matrix S whose elements are defined as follows.

$$s_{ij} = \max\{PMI(i, j), 0\} \tag{3.10}$$

The item embedding model for implicit feedback can be summarized as follows: (1) construct an item-item matrix S regarding the co-occurrence of items in the click history of users (the PPMI matrix), and (2) factorize matrix S to obtain the representations of items. The factorization can be performed following the PMF method that was described in Section 3.2.2.

3.3.2 Generative collaborative item embedding model

We have presented the idea of item embedding via factorizing the PPMI matrix regarding items from the implicit feedback. We are ready to present the idea to combine that model with the MF model for explicit feedback. Rather than performing two independent models: *matrix factorization* on user ratings, and *item embedding* on implicit feedback, we connect them into a unified model which is described below.

In item embedding above, item i will be represented by two vectors: *embedding vector* $\mathbf{w}_i$ and *context vector* $\mathbf{z}_i$ which are derived from the observations of the PPMI matrix. Additionally, the model adds an offset value to the embedding vector $\mathbf{w}_i$ to capture the deviation from the item vector learned from the implicit feedback. This deviation can be interpreted as the contribution of explicit feedback information into the representation of the items and is useful when the click information does not reflect the preference of a user to an item. For example, a user clicks an item many

times and finally found that he/she does not like the item, giving a low rating score.

The graphical model of the proposed model is displayed in Fig.3.2.

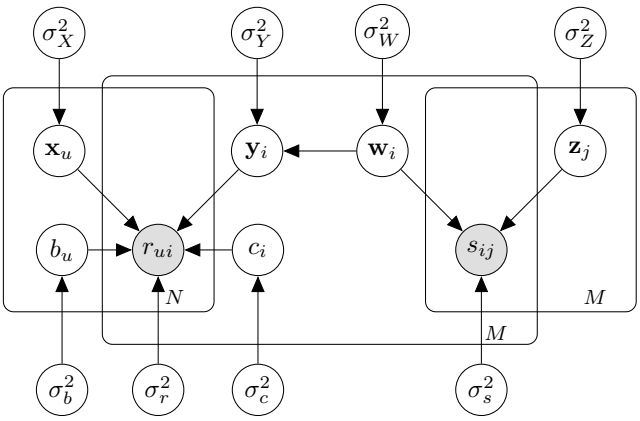


Figure 3.2: The graphical model of the proposed model.

The generative process for modeling the rating scores as well as the implicit feedback data is shown below.

1. **Item embedding model**

   (a) For each item i: draw the embedding vector $\mathbf{w}_i$ and context vector $\mathbf{z}_i$

$$\mathbf{w}_i \propto \mathcal{N}(0, \sigma_W^2 \mathbf{I}) \tag{3.11}$$

$$\mathbf{z}_i \propto \mathcal{N}(0, \sigma_Z^2 \mathbf{I}) \tag{3.12}$$

   (b) For each pair (i, j), draw s_{ij} of the PPMI matrix:

$$s_{ij} \propto \mathcal{N}(\mathbf{w}_i^\top \mathbf{z}_j, \sigma_S^2) \tag{3.13}$$

2. **Rating model**

   (a) For each user u: draw user vector $\mathbf{x}_u$ and bias term b_u

$$\mathbf{x}_u \propto \mathcal{N}(0, \sigma_X^2 \mathbf{I}) \tag{3.14}$$

$$b_u \propto \mathcal{N}(0, \sigma_b^2) \tag{3.15}$$

(b) For each item i: draw the item vector $\mathbf{y}_i$ and the bias term c_i

$$\mathbf{y}_i \propto \mathcal{N}(\mathbf{w}_i, \sigma_Y^2) \tag{3.16}$$
$$c_i \propto \mathcal{N}(0, \sigma_c^2) \tag{3.17}$$

(c) For each pair (u, i): draw the rating score

$$r_{ui} \propto \mathcal{N}(\mu + b_u + c_i + \mathbf{x}_u^\top \mathbf{y}_i, \sigma_R^2) \tag{3.18}$$

3.3.3 Parameter learning

The parameters of the model $\Theta = \{\mathbf{x}_{1:N}, \mathbf{y}_{1:M}, \mathbf{w}_{1:M}, \mathbf{z}_{1:M}, b_{1:N}, c_{1:N}\}$ are learned by maximizing the log posterior distribution which is equivalent to minimizing the following loss function.

$$\begin{aligned}\mathcal{L}(\Theta) = &\frac{1}{2}\sum_{(u,i)\in\mathcal{R}}[r_{ui} - (\mu + b_u + c_i + \mathbf{x}_u^\top \mathbf{y}_i)]^2 \\ &+ \frac{\lambda}{2}\sum_{(i,j)\in\mathcal{S}}(s_{ij} - \mathbf{w}_i^\top \mathbf{z}_j)^2 \\ &+ \frac{\lambda_X}{2}\sum_{u=1}^{N}||\mathbf{x}_u||_F^2 + \frac{\lambda_Y}{2}\sum_{i=1}^{M}||\mathbf{y}_i - \mathbf{z}_i||_F^2 \\ &+ \frac{\lambda_W}{2}\sum_{i=1}^{M}||\mathbf{y}_i||_F^2 + \frac{\lambda_Z}{2}\sum_{j=1}^{M}||\mathbf{z}_j||_F^2 \\ &+ \frac{\lambda_b}{2}\sum_{u=1}^{N} b_u^2 + \frac{\lambda_c}{2}\sum_{i=1}^{M} c_i^2\end{aligned} \tag{3.19}$$

where $\mathcal{S} = \{(i,j)|s_{ij} > 0\}, \lambda = \sigma_R^2/\sigma_S^2, \lambda_X = \sigma_R^2/\sigma_X^2, \lambda_Y = \sigma_R^2/\sigma_Y^2, \lambda_W = \sigma_R^2/\sigma_W^2$, and $\lambda_Z = \sigma_R^2/\sigma_Z^2$.

Function in Eq.(3.19) is not convex with respect to $\mathbf{x}_{1:N}$, $\mathbf{y}_{1:M}$, $\mathbf{w}_{1:M}$, $\mathbf{z}_{1:M}$, $b_{1:N}$, $c_{1:M}$, but it is convex if we keep five of them fixed. Therefore, it can be solved using alternative least square (ALS) method, similar to the method described in (Hu et al., 2008).

For each user u, at each iteration, we calculate the partial derivative of $\mathcal{L}(\Theta)$ with respect to $\mathbf{x}_u$ while fixing other parameters. By setting this derivative to be zero: $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_u} = 0$, we can obtain the update rule of $\mathbf{x}_u$ as in Eq.(3.20).

$$\mathbf{x}_u^{new} = \Big(\sum_{i \in \mathcal{R}_u} \mathbf{y}_i \mathbf{y}_i^\top + \lambda_X \mathbf{I}_d \Big)^{-1} \Big(\sum_{i \in \mathcal{R}_u} r_{ui} \mathbf{y}_i \Big) \tag{3.20}$$

where $\mathbf{I}_d$ is the d-dimensional identity matrix.

By doing the same way, we can obtain the update rules for $\mathbf{w}_i$, $\mathbf{z}_j$, b_u and c_i as follows.

$$\mathbf{y}_i^{new} = \Big(\sum_{u \in \mathcal{R}_i} \mathbf{x}_u \mathbf{x}_u^\top + \lambda_Y \mathbf{I}_d \Big)^{-1} \Big(\lambda_Y \mathbf{y}_i + \sum_{u \in \mathcal{R}_i} (r_{ui} - B_{ui}) \mathbf{x}_u \Big) \tag{3.21}$$

$$\mathbf{w}_i^{new} = \Big(\lambda \sum_{j \in \mathcal{S}_i} \mathbf{z}_j \mathbf{z}_i^\top + \lambda_\rho \mathbf{I}_d \Big)^{-1} \Big(\lambda_Y \mathbf{y}_i + \lambda \sum_{j \in \mathcal{S}_i} s_{ij} \mathbf{z}_j \Big) \tag{3.22}$$

$$\mathbf{z}_j^{new} = \Big(\lambda \sum_{i \in \mathcal{S}_j} \mathbf{y}_i \mathbf{y}_i^\top + \lambda_\alpha \mathbf{I}_d \Big)^{-1} \Big(\lambda \sum_{i \in \mathcal{S}_j} s_{ij} \mathbf{y}_i \Big) \tag{3.23}$$

$$b_u^{new} = \frac{\sum_{i \in \mathcal{R}_u} r_{ui} - (\mu |\mathcal{R}_u| + \sum_{i \in \mathcal{R}_u} c_i + \mathbf{x}_u^\top \mathbf{y}_i)}{|\mathcal{R}_u| + \lambda_b} \tag{3.24}$$

$$c_i^{new} = \frac{\sum_{u \in \mathcal{R}_i} r_{ui} - (\mu |\mathcal{R}_i| + \sum_{u \in \mathcal{R}_i} b_u + \mathbf{x}_u^\top \mathbf{y}_i)}{|\mathcal{R}_i| + \lambda_c} \tag{3.25}$$

where $B_{ui} = \mu + b_u + c_i$, $\mathcal{S}_i = \{j | s_{ij} > 0\}$, $\mathcal{S}_j = \{i | s_{ij} > 0\}$, $\mathcal{R}_u$, again, is the set of items that u gave ratings, and $\mathcal{R}_i$ is the set of users that gave ratings to i.

The method can be summarized as in the Algorithm 2.

Computational complexity. For user vectors, as analyzed in (Hu et al., 2008), the complexity for updating N users in an iteration is $\mathcal{O}(d^2 |\mathcal{R}| + d^3 N)$, where $|\mathcal{R}|$ is the number of non-zero entries of rating matrix R. Since $|\mathcal{R}| >> N$, if d is small, this complexity is a linear in the size of the input matrix. For item vector updating, we can also easily show that the running time for updating M items in an iteration is $\mathcal{O}(d^2(|\mathcal{R}| + |\mathcal{S}|) + d^3 M)$, where $|\mathcal{S}|$ is the number of non-zero entries of matrix S. We can see that the computational complexity linearly scales with the number of users and the number of items. Furthermore, this algorithm is easy to be parallelized to adapt to large scale data. For example, in updating user vectors $\mathbf{x}_u$, the update rule of user u is independent of other users' vectors, therefore, we can compute $\sum_i \mathbf{y}_i \mathbf{y}_i^\top$ in advance, and update $\mathbf{x}_u$ in parallel.

Algorithm 2: The ALS algorithm for the proposed model.

Input : Rating matrix: $\mathbf{R}$, PPMI matrix $\mathbf{S}$, regularization parameters: $\lambda_X, \lambda_Y, \lambda_W, \lambda_Z, \lambda_b, \lambda_c$

Output: The latent factor vectors $\mathbf{x}_{1:N}, \mathbf{y}_{1:M}, \mathbf{w}_{1:M}, \mathbf{z}_{1:M}$ and the biases $b_{1:N}, c_{1:N}$ Randomly initialize $\mathbf{x}_{1:N}, \mathbf{y}_{1:M}, \mathbf{w}_{1:M}, \mathbf{z}_{1:M}, b_{1:N}, c_{1:M}$ from Gaussian distributions

for *epoch=1 ... T* **do**
  for *u=1 ... N* **do**
    Update user latent factor vector $\mathbf{x}_u$ by Eq.(3.20)
    Update bias term b_u by Eq.(3.24)
  end
  for *i=1 ... M* **do**
    Update item embedding vector $\mathbf{w}_i$ by Eq.(3.22)
    Update item context vector $\mathbf{z}_i$ by Eq.(3.23)
    Update item vector $\mathbf{y}_i$ by Eq.(3.21)
    Update bias term c_i by Eq.(3.25)
  end
end

3.3.4 Rating prediction

After learning the model parameters $\Theta = \{\mathbf{x}_{1:N}, \mathbf{y}_{1:M}, \mathbf{w}_{1:M}, \mathbf{z}_{1:M}, b_{1:N}, c_{1:N}\}$, the proposed model can be used for predicting missing ratings. We consider two cases of rating predictions: **in-matrix** prediction and **out-of-matrix** prediction. In-matrix prediction refers to the case that we predict the rating of user u to item i, where i has not been rated by u but has been rated by at least one other users. Out-matrix refers to the case that we predict the rating of user u to item i, where i has not been rated by any users (i.e., i has implicit feedback only).

Let $\mathcal{D}$ be the observed data (observed rating scores), the unobserved r_{ui} can be estimated as in Eq.(3.26).

$$\begin{aligned} \mathbb{E}\big[r_{ui}|\mathcal{D}\big] &= \mathbb{E}\big[\mathbf{x}_u|\mathcal{D}\big]^\top \mathbb{E}\big[\mathbf{y}_i|\mathcal{D}\big] \\ \hat{r}_{ui} &\approx \mu + b_u + c_i + \mathbf{x}_u^\top \mathbf{y}_i \end{aligned} \tag{3.26}$$

3.4 Empirical study

3.4.1 Datasets

We use three public datasets in different domains with varying sizes. The datasets are:

MovieLens 1M (ML-1m): a dataset of user-movie ratings collected from MovieLens, an online film service. It contains 1 million ratings in the range $1-5$ to 4000 movies by 6000 users. The dataset is available at GroupLens[1]

MovieLens 20M (ML-20m): a dataset of user-movie ratings collected from MovieLens, an online film service. It contains 20 million ratings in the range $1-5$ to 27,000 movies by 138,000 users. The dataset is available at GroupLens[2]

Bookcrossing: A dataset collected by Cai-Nicolas Ziegler in August and September 2004 from the Book-Crossing[3]. The dataset contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit/implicit) about 271,379 books. We remove users and items that have no explicit feedback.

The statistical information about the datasets is given in Table 3.2.

Table 3.2: Statistical information of the datasets

	ML-1m	ML-20m	Bookcrossing
# of users	6,040	138,493	77,805
# of items	3,706	26,744	185,973
value of ratings	1 - 5	0.5 - 5	1–10
average rating	3.58	3.53	7.61
# of ratings	1,000,209	20,000,263	357,246
rating density (%)	4.47	0.53	0.0029
# of clicks	-	-	892,185
click density (%)	-	-	0.0062

Since Movielens datasets contain only explicit feedback, we artificially create the implicit feedback and explicit feedback data following (Bell and Koren, 2007). For the implicit feedback, we use all the rating data by considering whether a user rated an item or not. In other words, the implicit feedback is obtained by binarizing the rating data. For explicit feedback, we randomly pick 10%, 50%, and 90% of

[1]https://grouplens.org/datasets/movielens/1m/
[2]https://grouplens.org/datasets/movielens/20m/
[3]http://www.bookcrossing.com/

the original rating data and assume that only these amounts of ratings are available. These datasets are: ML1-10, ML1-50, ML1-90 (obtained by picking 10%, 50%, 90% from ML-1m, respectively) and ML20-10, ML20-50, ML20-90 (obtained by picking 10%, 50%, 90% from ML-20m, respectively). The densities of the rating matrices of these datasets are given in Table 3.3.

Table 3.3: Statistical information of some subsets drawn from Movielens data

	Original dataset	% selected	Density (%)
ML1-10	ML-1m	10%	0.3561
ML1-50	ML-1m	50%	1.6022
ML1-90	ML-1m	90%	2.8206
ML20-10	ML-20m	10%	0.1001
ML20-50	ML-20m	50%	0.2108
ML20-90	ML-20m	90%	0.3459

3.4.2 Evaluation

We split the rating data into two parts: 80% for the training set and 20% for as ground-truth for testing. From the training set, we randomly pick 10% as a validation set that will be used for model selection and checking stopping condition of the training phase. In evaluating the in-matrix prediction, when splitting data, we make sure that all the items in the test set appear in the training set (to ensure that all the items in the test set have at least one rating in the past). In evaluating out-of-matrix prediction, we make sure that none of the items in the test set appear in the training set (to ensure that none of the items in the test set have any rating in the past).

The model is trained on the training dataset and the optimal parameters are obtained by using the validation set. The model with these optimal parameters is then used to predict ratings for user-item pairs that appear in the test set. We use Root Mean Square Error (RMSE), as the metric to measure the performance of the models. RMSE measures the deviation between the rating predicted by the model and the true ratings (given by the test set) and is defined as follows.

$$RMSE = \sqrt{\frac{1}{|Test|} \sum_{(u,i) \in Test} (r_{ui} - \hat{r}_{ui})^2} \tag{3.27}$$

where $|Test|$ is the size of the test set. The smaller the value of RMSE on the test

set is, the better the performance of the model is.

3.4.3 Competing methods

For in-matrix prediction. We compare our method with the following three factorization models:

1. *PMF* (Salakhutdinov and Mnih, 2008): a state-of-the-art method for rating prediction which is described in Section 3.2.2.

2. *NMF* (non-negative matrix factorization) (Lee and Seung, 2000): a factorization model with the constraint that all the components of user factors and item factors must be non-negative

3. *SVD++* (Koren, 2008): a factor model that exploits both explicit feedback and implicit feedback in rating prediction. The implicit feedback used in SVD++ is in the form: "who rated what?" which is inferred from the explicit feedback data.

We used the Librec [4], an open source library to run the competing methods above.

For out-of-matrix prediction. Since conventional collaborative filtering methods cannot predict ratings for the items with no prior ratings, we will use the *user-average* as the competing method. The user-average estimates the rating of a user to an item by the average of the ratings that he or she gave to other items.

3.4.4 Parameter settings

In all settings, we set the dimension of the latent space to $d = 20$. For PMF, NMF, and SVD++, we used grid-search to find the optimal values of the regularization terms that produce the best performance on the validation set. We found that, $\lambda_u = \lambda_u = 0.01$ give good performance. For our proposed method, we explored the parameters in different settings. First, we fixed $\lambda = 1$ and used grid search for finding the optimal values of the remaining parameters that give good performance on the validation set. We found that $\lambda_X = \lambda_Y = \lambda_W = \lambda_Z = \lambda_b = \lambda_c = 10$ give good performance and used this setting in comparing with the competing methods. Second, we explored different values of λ while fixing the remaining parameters in

[4] http://librec.net/

order to study how the contribution of implicit data affects the performance of the model. Third, we explored different values of λ_Y, which control the divergence of $\mathbf{y}_i$ from $\mathbf{w}_i$, in order to study how the performance of the model is influenced by the deviation of the item representation from the item embedding vector.

3.4.5 Experimental results

We report the RMSE on the test set of the in-matrix prediction for datasets ML1-50, ML20-50 and Bookcrossing, in Table 3.4. The test RMSE of in-matrix prediction show that our method outperforms the competing methods on over three datasets.

Table 3.4: Test RMSE of in-matrix prediction on three datasets.

Methods	Datasets		
	ML1-50	ML20-50	Bookcrossing
PMF	0.8983	0.8441	2.1663
NMF	1.0051	0.9891	1.9374
SVD++	0.8871	0.8191	1.6916
CoMF (our)	**0.8498**	**0.8024**	**1.6558**

We can observe that our method and SVD++ are much better than PMF and NMM, which use explicit feedback only. This indicates that exploiting implicit feedback is a key point to increase the prediction performance. Our method outperforms SVD++ is because we can exploit more information from implicit feedback data than SVD++ does. While SVD++ uses only "implicit feedback" data that is inferred from the explicit feedback, our method uses much more implicit feedback which comes from a different and independent source.

In Bookcrossing dataset, the test RMSE of NMF and PMF are far worse than SVD++ and our method. The reason is that the explicit feedback Bookcrossing data is extremely sparse (0.0029%) and explicit feedback only is not enough for prediction.

Table 3.5: Test RMSE of out-of-matrix prediction on three datasets.

Methods	Datasets		
	ML1-50	ML20-50	Bookcrossing
User-average	1.0431	0.9673	1.7142
CoMF (our)	**1.0132**	**0.9494**	**1.6828**

The test RMSE of out-of-matrix prediction for ML1-50, ML20-50, and Bookcrossing, is given in Table 3.5. The results show that our method outperforms imputing ratings by user-average. From the result, we can see that the performance of our method, in this case, is worse than in in-matrix prediction. This is reasonable because this prediction is almost entirely based on the implicit feedback data.

Effect of the sparseness of explicit data

We study the performances of the methods on different levels of sparsities of explicit feedback data.

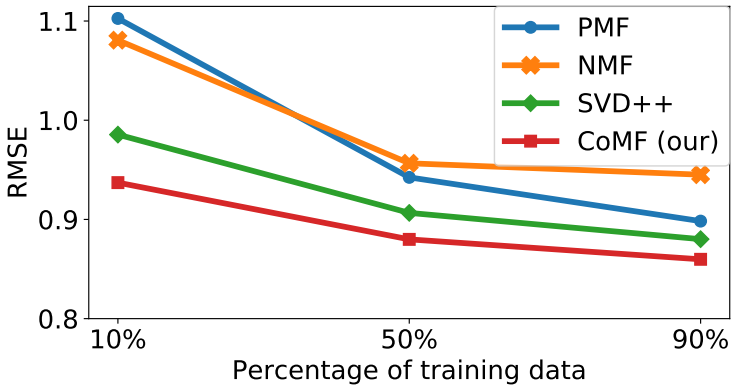
Figure 3.3 and Fig.3.4 show the test RMSE results of in-matrix and out-of-matrix prediction tasks on different subsets of the data. The experimental results show that our proposed method outperforms all competing methods in both in-matrix prediction and out-of-matrix prediction on two datasets over different levels sparsities of explicit feedback data. For all methods, the predicting accuracies increase with the density of explicit feedback. This is expected because the explicit feedback data is reliable for inferring users' preferences.

In all cases, the differences between our proposed method with the competing methods are most pronounced in the most sparse subsets (ML1-10 or ML20-10). This indicates the effectiveness of our proposed method for sparse data.

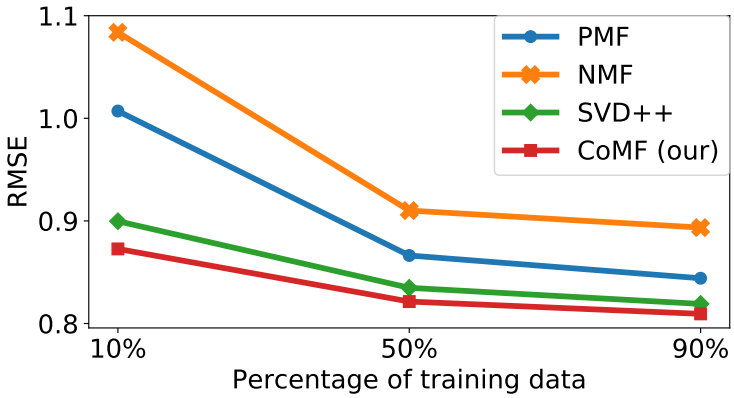
Is joint learning important?

In the proposed model, we train the matrix factorization component jointly with the item embedding component. This "end-to-end" fashion has shown the effectiveness in many machine learning tasks. Here, we are interested in studying the effectiveness of jointly training the two models. To do so, we compare the proposed model with the model where the item embedding and the matrix factorization are trained separately. First, we use item2vec (Barkan and Koenigstein, 2016) to compute the embedding vectors of the items. We then fixed the embedding vectors of the items and use them as the item latent factor vectors, and learn the user latent factor vectors. We name this model *Item2Vec+MF*.

The accuracies of the Item2Vec+MF and CoMF models are reported in Table 3.6a (in-matrix prediction) and Table 3.6b (out-of-matrix prediction). We can observe that the performance of Item2Vec+MF is much worse than the proposed model. This is



(a) ML-1m dataset



(b) ML-20m dataset

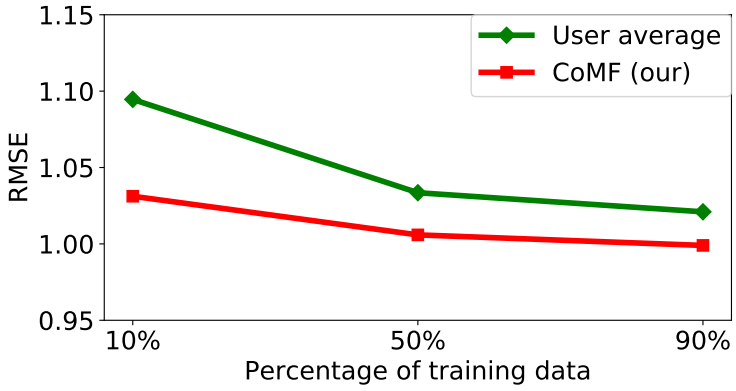
Figure 3.3: Test RMSE of in-matrix prediction for different subsets of ML-1m and ML-20m data

reasonable because the item embeddings learned by Item2Vec is well-suit for the rating prediction. In contrast, the proposed model is able to keep a balance between the contribution of the rating data and the click data in for learning item latent factor vectors that reflect both rating data and click data.

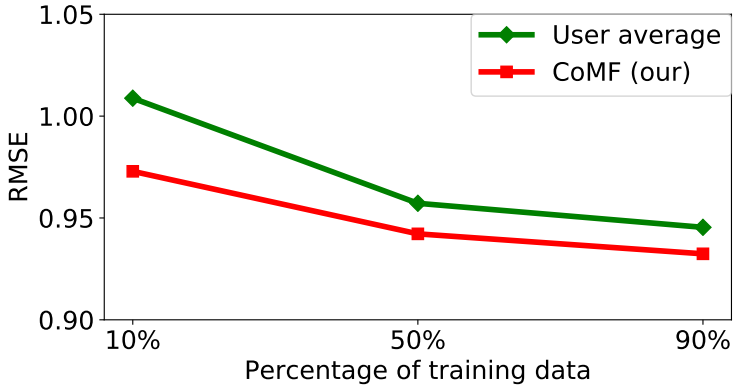
Impact of parameter λ

As in the Eq.(3.19), parameter λ controls the level of contribution of implicit feedback data to the model. If $\lambda = 0$, the model reduces to the original MF which uses explicit feedback data only for modeling users and items. If $\lambda = \infty$, the model uses only information from the implicit feedback to model items. In this part, we vary λ while fixing other parameters to study the effect of λ on the accuracy of the model.

Figure 3.5 shows the test RMSE of in-matrix prediction task of CoMF on the ML1-50 data when the λ is varied. From the result, we can observe that the predic-



(a) ML-1m dataset



(b) ML-20m dataset

Figure 3.4: Test RMSE of out-of-matrix prediction for different subsets of ML-1m and ML-20m data

tion performance is influenced significantly by the value of λ. For small values of λ, the test RMSE is relatively high, it decreases when λ increases. However, when λ goes over a certain threshold, the test RMSE starts increasing. This can be explained as follows. For a very small value of λ, the model mainly uses information from the explicit feedback which is too sparse to model the users and items. When the value of λ becomes very large, the model mainly uses the implicit feedback data for modeling the items, therefore, is not reliable. The best values of λ should balance the contribution of implicit and explicit feedback.

Impact of parameter λ_Y

λ_Y is the parameter that controls the deviation of item latent vector $\mathbf{y}_i$ from the item embedding vector $\mathbf{w}_i$. When λ_Y is small, the value of $\mathbf{y}_i$ is allowed to diverge from $\mathbf{w}_i$; in this case, the information for modeling item i mainly comes from explicit

Table 3.6: Comparison between joint learning (CoMF) and separate learning (Item2Vec+MF).

(a) In-matrix prediction

Methods	Datasets		
	ML1-50	ML20-50	Bookcrossing
Item2Vec+MF	0.8984	0.8355	1.9014
CoMF (our)	**0.8498**	**0.8024**	**1.6558**

(b) Out-of-matrix prediction

Methods	Datasets		
	ML1-50	ML20-50	Bookcrossing
Item2Vec+MF	1.0390	0.9784	1.7027
CoMF (our)	**1.0132**	**0.9494**	**1.6828**

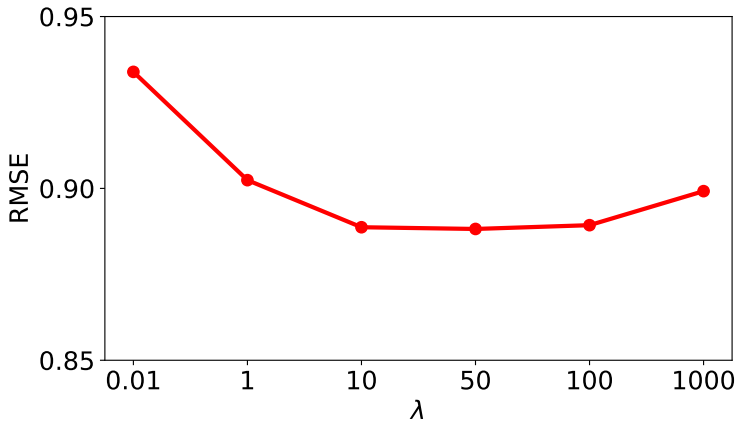


Figure 3.5: Test RMSE of in-matrix prediction task on ML1-50 dataset corresponding to different values of λ.

feedback. On the other hand, when λ_Y increases, $\mathbf{y}_i$ becomes closer to $\mathbf{w}_i$; in this case, the item vectors mainly come from the embedding model of implicit feedback.

Figure 3.6 shows the test RMSE of in-matrix prediction task of CoMF on the ML1-50 dataset when λ_Y is varied while other parameters are fixed. From the result, we can observe that, for small values of λ_Y, the model produces low prediction accuracy (high test RMSE). The reason is when λ_Y is small, the model mostly relies on the explicit feedback which is very sparse and can not model users and items well. When λ_Y increase, the model starts using implicit feedback for prediction, the accuracy will increase. However, when λ_Y reaches a certain threshold, the accuracy starts decreasing. This is because when λ_Y is too large, the representations of items mainly come from implicit feedback, and therefore the model becomes less reliable

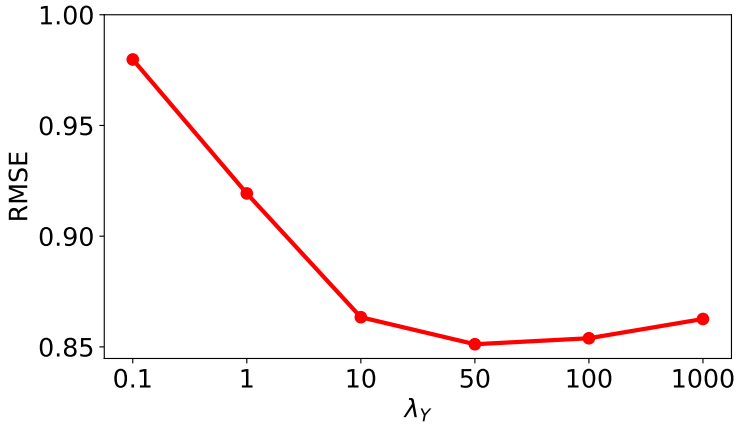


Figure 3.6: Test RMSE of in-matrix prediction task on ML1-50 dataset corresponding to different values of λ_Y.

to model the rating data.

3.5 Chapter Summary

In this section, we proposed a probabilistic model that combines explicit feedback and implicit feedback for rating prediction task. The model is a combination of two models: MF for explicit feedback and item embedding for implicit feedback. The experimental results showed that our proposed method improved the accuracy of rating prediction for three real-world datasets. Our method also can efficiently learn the latent representations of items whose rating data is not available and efficiently predict the missing ratings for them.

There are several ways to extend or improve this work. In this model, the optimal hyper-parameters are found by performing grid search. It is very time consuming for large datasets or if we increase the range of choices for hyper-parameters. One direction to improve this work is to develop a fully Bayesian model that treats hyper-parameters as random variables and we can perform inferring posterior distribution over hyper-parameters. Another direction is to inject the auxiliary information of the items (e.g., textual data or visual data) into the model for addressing the cold-start problem.

Chapter 4

NPE: Neural Personalized Embedding

Matrix factorization is one of the most efficient approaches in recommender systems. However, such algorithms, which rely on the interactions between users and items, perform poorly for “cold-users" (users with a little history of such interactions) and at capturing the relationships between closely related items. To address these problems, we propose a neural personalized embedding (NPE) model, which improves the recommendation performance for cold-users and can learn effective representations of items. It models a user’s click to an item in two terms: the *personal preference* of the user for the item, and the *relationships* between this item and other items clicked by the user. We show that NPE outperforms competing methods for top-N recommendations, especially for cold-user recommendations. We also performed a qualitative analysis that shows the effectiveness of the representations learned by the model.

4.1 Introduction

As motivated in Chapter 1, in order to make good recommendations, it is essential to understand the preferences of users to items. In this chapter, we focus on the recommendation with implicit data. Given previous click/purchase data, we are interesting in making a list of n items to suggest to a specific user (the top-n recommendation).

In this setting, matrix factorization (MF) is one of the most efficient approaches which find the latent representations of users and items (Hu et al., 2008; Pan et al.,

2008). To address the sparseness of the user-item matrix, additional data are integrated into MF as "side information". This might include textual information for article recommendations (Wang and Blei, 2011; Wang et al., 2015a), product images in e-commerce (He and McAuley, 2016), or music signals for song recommendations (Oord et al., 2013). However, there are two major issues with these MF-based algorithms. First, these models are poor at modeling *cold-users* (i.e., users who have only a short history of relevant activities). Second, because these models consider only user-item interactions, the item representations poorly capture the relationships among closely related items (Koren, 2009).

One approach to the cold-user recommendation is to exploit user profiles. Such proposed models (Li et al., 2015b; Tang and Liu, 2017) can learn user representations from their profiles (e.g., gender and age). In this way, these models can make recommendations to new users who have no historical activities, provided their user profiles are available. However, user profiles are often very noisy, and in many cases, they are simply not available. Another approach is item-similarity based models (Linden et al., 2003; Sarwar et al., 2001), which recommends items based on item-item similarity. The main issue of this approach is that it considers only the most recent click when making a recommendation, ignoring previous clicks. In addition, these models are not personalized.

In item representations learning, Item2Vec (Barkan and Koenigstein, 2016) is an efficient model that borrows the idea behind word-embedding techniques (Mikolov et al., 2013b) for learning item representations. However, the main goal of Item2Vec is to learn item representations and it cannot be used directly for predicting missing entries in a user-item matrix. Furthermore, in making recommendations, Item2Vec is not personalized: it recommends items based on the similarities between items, computed using item representations, and ignores users' historical activities.

To address these problems, we propose a neural personalized embedding (NPE) model that fuses item relationships for learning effective item representations in addition to improving recommendation quality for cold-users. NPE models a user's click on an item by assuming that there are two signals driving the click: the *personal preference* of the user with respect to the item and the *relationships* between this item and other items that the user has clicked.

To model the *personal preference* term, we adopt the same approach as MF, which views the preference of a user for an item as the inner product of the corre-

sponding factor vectors. To model the *relationships* among items, we propose an *item-embedding* model that generalizes the idea behind word-embedding techniques to click data. However, our item-embedding model differs from the word-embedding model in that the latter can only learn word representations. In contrast, our embedding model can both learn item representations and fill in the user-item matrix simultaneously. The experimental results demonstrate that the proposed method outperforms the competing methods on top-N recommendation task across all datasets.

The rest of this chapter is organized as follows. In Section 4.2 we present in detail NPE, the proposed model. Section 4.3 describes the experiments and report the experimental results. In Section 4.4, we summarize the proposed model and discuss about the future directions for extending this work.

4.2 NPE: Neural Personalized Embedding

We propose NPE, a factor model that explains users' clicks by capturing the preferences of users for items and the relationships between closely related items. We will describe the model and how to learn the model parameters.

4.2.1 Problem Formulation

Each entry $r_{u,i}$ in the user-item preference matrix $\mathbf{R}$ has one of two values 0 or 1, such that $r_{u,i} = 1$ if user u has clicked item i and $r_{u,i} = 0$ otherwise. We assume that $r_{u,i} = 1$ indicates that user u prefers i, whereas $r_{u,i} = 0$ indicates that this entry is non-observed (i.e., a missing entry).

Given a user u and the set of items that u previously interacted, our goal is to predict a list of items that u may find interesting (top-N recommendations).

The notations used in this section are defined in Table 4.1.

4.2.2 Model Formulation

We denote the observations for user u as:

$$\mathbf{r}_u = (r_{u,1}, r_{u,2}, \ldots, r_{u,M}). \tag{4.1}$$

NPE models the probability of each observation conditioned on user u and its

Table 4.1: The notations used throughout the Chapter 4.

Notation	Meaning
N, M	the number of users and items, respectively
$\mathbf{R}$	the user-item matrix (e.g., click matrix)
$\mathbf{r}_u$	the observation data for user u (i.e., the row corresponding to user u of matrix $\mathbf{R}$)
d	the dimensionality of the embedding space
H	the dimensionality of the user input vector
L	the dimensionality of the item input vector
$\mathbf{x}_u$	the input vector of user u, $x_u \in \mathbb{R}^H$
$\mathbf{y}_i$	the input vector of item i, $y_i \in \mathbb{R}^L$
$\mathbf{H}$	the user embedding matrix, $\mathbf{H} \in \mathbb{R}^{H \times D}$
$\mathbf{W}$	the item-embedding matrix, $\mathbf{W} \in \mathbb{R}^{L \times D}$
$\mathbf{V}$	the item context matrix, $\mathbf{V} \in \mathbb{R}^{L \times D}$
$\mathbf{h}_u$	the embedding vector of user u, $\mathbf{h}_u \in \mathbb{R}^D$
$\mathbf{w}_i$	the embedding vector of item i, $\mathbf{w}_i \in \mathbb{R}^D$
$\mathbf{v}_i$	the context vector of item i, $\mathbf{v}_i \in \mathbb{R}^D$
$\mathbf{\Theta}$	The set of all model parameters
$\Omega(.)$	The regularization term
$\mathbf{c}_{u,i}$	the set of items that user u clicked, excluding i (the *context* items)
n	the negative sampling ratio
$\mathcal{D}^+$	the set of positive examples, $\mathcal{D}^+ = \{(u,i)\|r_{u,i} = 1\}$
$\mathcal{D}^-$	the set of negative examples, which is obtained by sampling from zero entries of matrix $\mathbf{R}$

context items as:

$$p(r_{u,i} = 1|u, \mathbf{c}_{u,i}), \tag{4.2}$$

This equation captures the intuition behind the model, namely that the conditional distribution of whether user u clicks on item i is governed by two factors: (1) the personal preference of user u for item i, and (2) the set of items that u has clicked (i.e., $\mathbf{c}_{u,i}$).

The likelihood function for the entire matrix $\mathbf{R}$ is then formulated as:

$$p(\mathbf{R}) = \prod_{u=1}^{N} \prod_{i=1}^{M} p(r_{u,i}|u, \mathbf{c}_{u,i}). \tag{4.3}$$

The conditional probability expressed in Eq.(4.2) is implemented by a neural network. This neural network connects the input vectors of user u, item i, and context items $\mathbf{c}_{u,i}$ to their hidden representations as:

$$\mathbf{h}_u = \mathbf{f}(\mathbf{x}_u^\top \mathbf{H}), \tag{4.4}$$

$$\mathbf{w}_i = \mathbf{f}(\mathbf{y}_i^\top \mathbf{W}), \tag{4.5}$$

$$\mathbf{v}_{\mathbf{c}_{u,i}} = \mathbf{f}(\sum_{j \in \mathbf{c}_{u,i}} \mathbf{y}_i^\top \mathbf{V}), \tag{4.6}$$

where $\mathbf{f}(.)$ is an activation function such as ReLU.

Note that there are two hidden representations associated with item i: the *embedding vector* $\mathbf{w}_i$ and the *context vector* $\mathbf{v}_i$, which have different roles. $\mathbf{w}_i$ accounts for the attributes of item i, whereas $\mathbf{v}_i$ accounts for specifying the items that appear in its context.

We can then define the conditional probability in Eq.(4.2) via the hidden representations as:

$$p(r_{u,i} = 1 | u, \mathbf{c}_{u,i}) = \sigma(\mathbf{h}_u^\top \mathbf{w}_i + \mathbf{w}_i^\top \mathbf{v}_{\mathbf{c}_{u,i}}). \tag{4.7}$$

Note that the $\sigma(.)$ function on the right side of Eq.(4.7) comprises two terms: the first term $\mathbf{h}_u^\top \mathbf{w}_i$ accounts for how user u prefers item i, while the second term $\mathbf{w}_i^\top \mathbf{v}_{\mathbf{c}_{u,i}}$ accounts for the compatibility between item i and the items that u has already clicked.

From Eq.(4.7), we can also obtain the probability that $r_{u,i} = 0$ as:

$$p(r_{u,i} = 0 | u, \mathbf{c}_{u,i}) = 1 - \sigma(\mathbf{h}_u^\top \mathbf{w}_i + \mathbf{w}_i^\top \mathbf{v}_{\mathbf{c}_{u,i}}) \tag{4.8}$$

The conditional probability functions in Eq.(4.7) and Eq.(4.8) can be summarized in a single conditional probability function as:

$$p(r_{u,i} = r | u, \mathbf{c}_{u,i}) = \begin{cases} \hat{\mu}_{u,i}, & \text{if } r = 1, \\ 1 - \hat{\mu}_{u,i}, & \text{if } r = 0, \end{cases} \tag{4.9}$$

where $\hat{\mu}_{u,i} = \sigma(\mathbf{h}_u^\top \mathbf{w}_i + \mathbf{w}_i^\top \mathbf{v}_{\mathbf{c}_{u,i}})$.

4.2.3 The Model Architecture

The architecture of NPE is shown in Fig.4.1 as a multi-layer neural network. The first layer is the input layer which specifies the input vectors of (1) a user u, (2) a candidate item i, and (3) the context items. Above this is the second layer (the embedding layer), which connects to the input layer via connection matrices $\mathbf{H}$, $\mathbf{W}$, and $\mathbf{V}$. Above the embedding layer, two terms are calculated: the *personal preference* of user u for item i and the *relationship* between i and the context items. Finally, the model combines these two terms to compute the output, which is the probability that u will click i.

Note that, the input layer accepts a wide range of vectors that describe users and items such as one-hot vectors or content feature vectors obtained from side information. With such generic input vectors, our method can address the cold-start problem by using content feature vectors as input vectors for users and items. Since this work focuses on the pure collaborative filtering setting, we use only the identities of users and items in the form of one-hot vectors as input vectors. Investigating the effectiveness of using content feature vectors, is left for future work.

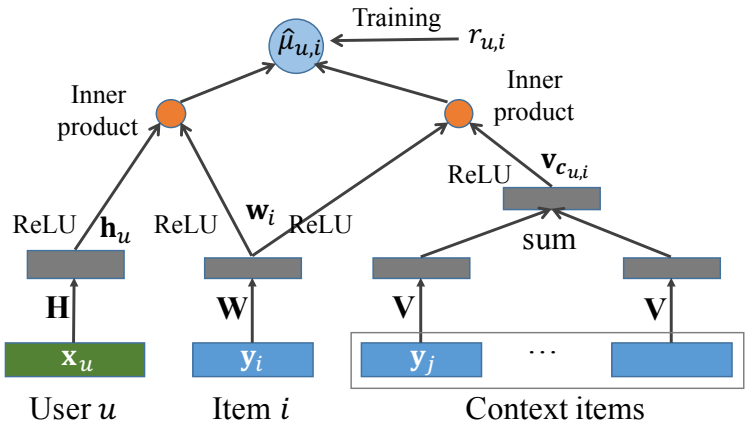


Figure 4.1: The architecture of NPE.

4.2.4 Objective Function

Given an observed matrix $\mathbf{R}$, our goal is to learn the model parameters Θ ($\mathbf{H}$, $\mathbf{W}$, $\mathbf{V}$) that maximize the likelihood function in Eq.(4.3). However, instead of modeling all zero entries, we only model a small subset of such entries by picking them randomly

(negative sampling). This gives:

$$p(\mathbf{R}) = \prod_{(u,i)\in\mathcal{D}^+} p(r_{u,i}|u, \mathbf{c}_{u,i}) \prod_{(u,i)\in\mathcal{D}^-} p(r_{u,i}|u, \mathbf{c}_{u,i}). \tag{4.10}$$

Maximizing the likelihood in Eq.(4.10) is equivalent to minimizing the following loss function (its negative log function):

$$\begin{aligned}\mathcal{L}(\mathbf{\Theta}) = &- \sum_{(u,i)\in\mathcal{D}^+} \log \hat{\mu}_{ui} - \sum_{(u,i)\in\mathcal{D}^-} \log(1 - \hat{\mu}_{ui}) \\ &+ \lambda\Omega(\mathbf{\Theta}),\end{aligned} \tag{4.11}$$

where $\hat{\mu}_{u,i} = \sigma(\mathbf{h}_u^\top \mathbf{w}_i + \mathbf{w}_i^\top \mathbf{v}_{\mathbf{c}_{u,i}})$.

This loss function is known as the *binary cross-entropy*.

4.2.5 Model Training

The model parameters are updated by the back propagation. We adopt the Adam (a mini-batch stochastic gradient descent approach) (Kingma and Ba, 2015) as the optimization algorithm for the back propagation. We do not perform negative sampling in advance, which can only produce a fixed set of negative samples. Instead, we perform negative sampling with each epoch, which enables diverse sets of negative examples to be used. The algorithm is summarized in Algorithm 3.

4.2.6 Connections with Previous Models

NPE vs. MF

In the conditional probability in Eq.(4.7), we can see that the $\sigma(.)$ function is a combination of two terms: (1) user preference and (2) item relationship. If the second term is removed, NPE will reduce to an original MF method.

NPE vs. Word Embedding

Similarly, if we remove the first element of $\sigma(.)$ in Eq.(4.7), NPE will model only the relationship among items. If we view each item as a word, and the set of items that a user clicked as a sentence, the model becomes similar to a word-embedding model. However, our embedding model differs in that word-embedding techniques

Algorithm 3: The training process using back propagation of the NPE model.

Input :

- $\mathbf{R}$: User-item preference matrix
- n: number of negative samples per positive example

Output: $\mathbf{\Theta} = \{\mathbf{H}, \mathbf{W}, \mathbf{V}\}$

Initialization: sample $\mathbf{H}, \mathbf{W}, \mathbf{V}$ from Gaussian distributions
for *epoch=1* $\ldots T$ **do**
    Sample negative examples $\mathcal{D}^-$
    $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$
    $\mathcal{O} = \text{Shuffle}(\mathcal{D})$
    for *t=1* $\ldots$ *# of mini-batches* **do**
        $\mathcal{B} = \text{next-mini-batch}(\mathcal{O})$
        $\text{Backprop}(\mathbf{\Theta}, \mathcal{B})$
    end
end

can only learn word (item) representations and cannot fill the user-item matrix directly. In contrast, our embedding model can learn effective item representations while predicting the missing entries in the user-item matrix.

4.3 Empirical Study

We have studied the effectiveness of NPE both quantitatively and qualitatively. In our quantitative analysis, we compared NPE with state-of-the-art methods on top-N recommendation task, using real-world datasets. We also performed a qualitative analysis to show the effectiveness of the item representations.

4.3.1 Datasets

We used three real-world datasets whose sizes varied from small to large-scale, from different domains.

- **Movielens 10M** (ML-10m): a dataset of user-movie ratings collected from MovieLens, an online film service. It contains 10 million ratings to 10,000 movies by 72,000 users. The dataset is available at GroupLens[1].

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

- **OnlineRetail** (Chen et al., 2012a): a dataset of online retail transactions provided at the UCI Machine Learning Repository[2]. It contains about 20,059 shopping transactions from December 1, 2010 to December 9, 2011 for a UK-based online retailer.

- **TasteProfile** [3]: a dataset of counts of song plays by users, as collected by Echo Nest.[4].

Table 4.2: Statistical information about the datasets.

	ML-10m	OnlineRetail	TasteProfile
#users	58,059	3,705	211,830
#items	8,484	3,644	22,781
#clicks	3,502,733	235,472	10,054,204
% clicks	0.71%	1.74%	0.21%

4.3.2 Experiment Setup

Data Preparation

For the ML-10m, we binarized the ratings, thresholding at 4 or above; for TasteProfile and OnlineRetail, we binarized the data and interpreted them as implicit feedback. Statistical information about the datasets is given in Table 4.2.

We partitioned the data into three subsets, using 70% of the data as the training set, 10% as the validation set, and the remaining 20% as the test set (ground truth).

Evaluation Metrics

After training the models on the training set, we evaluated the accuracy of their top-N recommendations using the test set. We used the rank-based metrics Recall@n and nDCG@n, which are common metrics in information retrieval, for evaluating the accuracy of the top-N recommendations. (We did not use “Precision" because it is difficult to evaluate, given that a zero entry can imply either that the user does not like the item or does not know about the item).

[2]https://archive.ics.uci.edu/ml/datasets/Online+Retail

[3]"The Instacart Online Grocery Shopping Dataset 2017", Accessed from https://www.instacart.com/datasets/grocery-shopping-2017

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

Competing Methods

We compared NPE with the following competing methods:

- Bayesian personalized ranking (**BPR**) (Rendle et al., 2009): an algorithm that optimizes the MF model with a pair-wise ranking loss
- Neural collaborative filtering (**NeuCF**) (He et al., 2017): a generalization of an MF method in which the inner product of user and item feature vectors are replaced by a deep neural network
- Sparse linear model (**SLIM**) (Ning and Karypis, 2011): a state-of-the-art method for top-N recommendations, which is based on the similarities between items.

4.3.3 Implementation Details

We implemented the proposed model using Pytorch. In the experiments, we use the embedding sizes $d \in \{8, 16, 32, 64, 128, 256\}$ and the negative sampling ratio $n \in \{1, 2, 4, 5, 8, 12, 16, 20\}$ to study the impact of the embedding size d and the negative sampling ratio n on the accuracies of the model. Since neural networks are prone to over-fitting, we apply a dropout after the hidden representation layer. We set the dropout rate 0.3 for all datasets. The weights for the connection matrices $\mathbf{H}$, $\mathbf{W}$, and $\mathbf{V}$ are initialized by Gaussian distributions with mean $\mu = 0$ and standard deviation $std = 0.01$. The size of each mini-batch was 10,000. The learning rate of the optimization algorithm is $lr = 0.01$, the weight decay is 0.01. We use early stopping to terminate the training process if the loss function does not decrease on the validation set for five epochs.

4.3.4 Experimental Results

Top-N Recommendations

Table 4.3 summarizes the Recall@20 and nDCG@20 for each model. Note that NPE significantly outperforms the other competing methods across all datasets for both Recall and nDCG. We emphasize that all methods used the same data. However, NPE benefits from capturing the compatibility between each item and other items picked by the same users.

Table 4.3: Recall and nDCG for three datasets, with embedding size $d = 64$ and negative sampling ratio $n = 4$.

Methods	ML-10m		OnlineRetail		TasteProfile	
	Re@20	nDCG@20	Re@20	nDCG@20	Re@20	nDCG@20
SLIM	0.1342	0.1289	0.2085	0.1015	0.1513	0.1422
BPR	0.1314	0.1253	0.2137	0.0943	0.1598	0.1398
NeuCF	0.1388	0.1337	0.2199	0.0911	0.1609	0.1471
NPE (our)	**0.1497**	**0.1449**	**0.2296**	**0.1742**	**0.1788**	**0.1594**

In Table 4.4, we summarize Recall@20 values for the four methods when different numbers of items were to be recommended. From these results, we can see that NPE consistently outperformed the other methods at all settings. The differences between NPE and the other methods are more pronounced for small numbers of recommended items. This is a desirable feature because we often only a consider a small number of top items (e.g., top-5 or top-10).

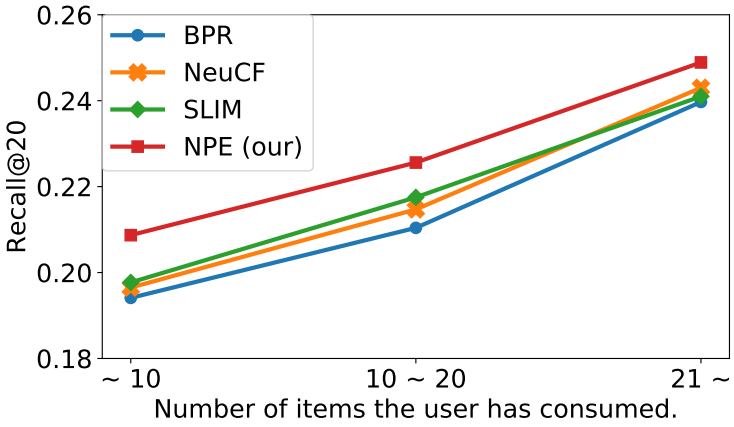
Table 4.4: Recall for different numbers of items to be recommended, with embedding size $d = 64$ and negative sampling ratio $n = 4$.

Methods	ML-10m			OnlineRetail			TasteProfile		
	Re@5	Re@10	Re@20	Re@5	Re@10	Re@20	Re@5	Re@10	Re@20
SLIM	0.1284	0.1298	0.1342	0.0952	0.1311	0.2085	0.1295	0.1304	0.1513
BPR	0.1254	0.1261	0.1314	0.0859	0.1222	0.2137	0.1307	0.1311	0.1598
NeuCF	0.1347	0.1363	0.1388	0.0871	0.1274	0.2199	0.1342	0.1356	0.1609
NPE (our)	**0.1451**	**0.1487**	**0.1497**	**0.1392**	**0.1667**	**0.2296**	**0.1428**	**0.1523**	**0.1788**

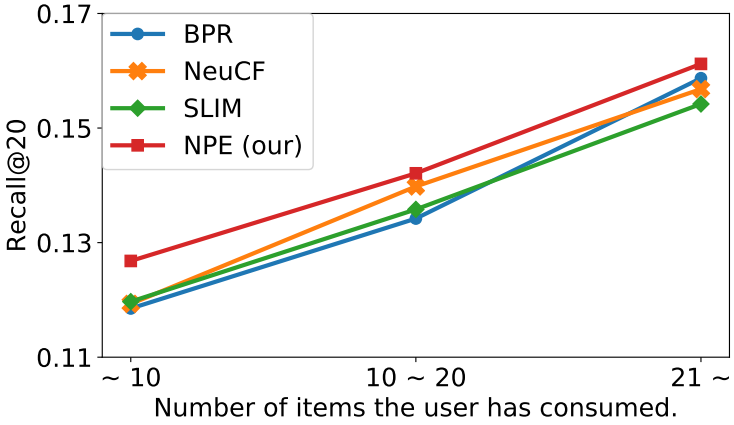
The Performance on Cold-Users

We studied the performance of the models for users who had few historical activities. To this end, we partitioned the test cases into three groups, according to the number of clicks that each user had. The *Low* group's users had less than 10 clicks, the *Medium* group's users had $10 \sim 20$ clicks, and the *High* group's users had more than 20 clicks.

Fig.4.2 shows the breakdown of Recall@20 in terms of user activity in the training set for the ML-10m and OnlineRetail. Although the details varied across datasets, the NPE model outperformed the other methods for all three groups of users. The differences between NPE and the other methods are much more pronounced for users who have fewest clicks. This is to be expected because, for such users, NPE captures the item relations when making recommendations.



(a) OnlineRetail dataset



(b) ML-10m dataset

Figure 4.2: Recall@20 for different groups of users

Effectiveness of the Item Representations

We evaluated the effectiveness of item representations by investigating how well the representations capture the *item similarity* and items that are often *purchased together*.

Similar items: The similarity between two items is defined as the cosine distance between their embedding vectors. Fig.4.3 shows three examples of the top-5 most similar items to a given item in the OnlineRetail dataset. We can see that the items' embedding vectors effectively capture the similarity of the items. For example, in the first row, given a *red alarm clock*, four of its top-5 similar items are also alarm clocks.

Items that are often purchased together: NPE can also identify items that are often purchased together. To assess if two items are often purchased together, we

Figure 4.3: Top-5 similar items for a given item. In each row, the given item is at the left and the top-5 similar items are to its right.

calculate the inner product of one item's embedding vector $\mathbf{w}_i$ and the other's context vector $\mathbf{v}_j$. A high value of this inner product indicates that these two items are often purchased together. Fig.4.4 shows an example of items that tend to be purchased together with the given item. Here, we see that buying a *knitting Nancy*, a child's toy, might accompany the purchase of other goods for children or for a household.

Figure 4.4: Top-5 items that are likely to be bought together with a given item. The given item is at the left and its top-5 most similar items are to its right.

Sensitivity Analysis

We also studied the effect of the hyper-parameters on the models' performance.

Impact of the embedding size: To evaluate the effects of the dimensionality of the embedding space on the top-N recommendations, we varied the embedding dimension d while fixing the other parameters. Table 4.5 summarizes the Recall@20 for NPE on the three datasets for various embedding sizes: $d = \{8, 16, 32, 64, 128, 256\}$. We can see that the larger embedding sizes seem to improve the performance of the models. The optimal embedding size for OnlineRetail is $d = 64$ and, for ML-10m and TasteProfile is $n = 128$.

Impact of the negative sampling ratio: During the training of NPE, we sampled negative examples. We studied the effect of the negative sampling ratio n on the

Table 4.5: Recall@20 for various embedding sizes, with negative sampling ratio $n = 4$.

d	ML-10m Re@20	OnlineRetail Re@20	TasteProfile Re@20
8	0.1428	0.1187	0.0987
16	0.1451	0.1596	0.1142
32	0.1441	0.1950	0.1509
64	**0.1497**	**0.2296**	0.1788
128	0.1482	0.2284	**0.1992**
256	0.1459	0.2248	0.1985

performance of NPE by fixing the embedding size $d = 32$ and evaluating Recall@20 for $n = \{1, 2, 4, 5, 8, 12, 16, 20\}$. From Table 4.6, we note that when n increases, the performance also increases up to a certain value of n. The optimal negative sampling ratios are $n = \{4, 5\}$ for OnlineRetail and $n = 8$ for ML-10m and TasteProfile. This is reasonable because ML-10m and TasteProfile, being larger than OnlineRetail, will need more negative examples.

Table 4.6: Recall@20 for different negative sampling ratios, with a fixed embedding size $d = 32$.

n	ML-10m Re@20	OnlineRetail Re@20	TasteProfile Re@20
1	0.1392	0.1608	0.1243
2	0.1418	0.1795	0.1451
4	0.1441	0.1950	0.1509
5	0.1478	**0.1952**	0.1585
8	**0.1563**	0.1941	**0.1621**
12	0.1531	0.1937	0.1615
16	0.1524	0.1925	0.1603
20	0.1496	0.1908	0.1598

4.4 Chapter Summary

We propose NPE, a neural personalized embedding model for collaborative filtering, is effective in making recommendations to cold-users and for learning item representations. Our experiments have shown that NPE can outperform competing methods with respect to top-N recommendations in general, and to cold-users in particular.

Our qualitative analysis also demonstrated that item representations can capture effectively the different kinds of relationships between items.

One future direction will be to study the effectiveness of the model when using available side information about items (i.e., use vector representations of item instead of one-hot vectors as input vector of the items). We also aim to investigate different negative sampling methods for dealing with zero values in the user-item matrix.

Chapter 5

Learning Product Representations from Shopping Transactions

Shopping transaction analysis is significant in understanding the shopping behaviors of customers. Existing models such as association rules are poor at modeling products which have short purchase histories and cannot be applied to new products (the cold-start problem). In this work, we propose BASTEXT, an efficient model of shopping baskets and the texts associated with the products (e.g., product titles). The model's goal is to learn the product representations from the textual contents, that can capture the relationships between the products in the baskets. Given the products already in a basket, a classifier identifies whether a potential product is relevant to the basket or not, based on their vector representations. This enables us to learn high-quality representations of the products. The experiments demonstrate that BASTEXT can efficiently model millions of baskets and that it outperforms the state-of-the-art methods in the next product recommendation task. Besides, we will also show that BASTEXT is a strong baseline for keyword-based product search.

5.1 Introduction

With the rapid development of the internet and online shopping services (e.g., *Amazon*, *Google play*), modern consumers are able to access a huge amount of products. Enabling customers to make sense of a huge amount of products is a big challenge. During the interaction with the system, consumers leave footprints, e.g., *click data* (i.e., product views) or *purchase data*. Such data is valuable in developing recom-

mender systems that can suggest products that meet the needs of customers. Traditional recommender systems such as collaborative filtering (CF) (Hu et al., 2008; Pan et al., 2008; Salakhutdinov and Mnih, 2008) recommend products based on user's long-term preferences without paying attention to the current shopping context.

In this work, we are interested in analyzing shopping transaction data. A shopping transaction, also known as a *shopping basket* or a *basket*, is a set of products that a customer buys on a single shopping trip. Such data could help reveal the relationships between products, understand the shopping behaviors, which are keys to make recommendations in a given context. For example, when an online-shopping customer is examining a mobile phone case, it is better to recommend him or her some other mobile phone cases, or other accessories such as screen protectors or headphones. It does not make sense to show him or her a *t-shirt* in that context.

A common approach to shopping basket analysis is *association-rules* (Agarwal et al., 1994), which discovers the rules in the form: "*Consumers who buy diapers are likely to buy baby food*". These rules can be formulated as $B => x$, where B is a set of products and x is another product. Such rules are effective in suggesting a customer next products to buy, given his current basket B. However, in a system with a large number of products, many relevant products have never co-occurred in any baskets. The relationships between such products are not able to be discovered by association rules. An approach to the context-based recommendation is neighborhood-based methods (Linden et al., 2003; Sarwar et al., 2001), which rely on the similarities between products (calculated from the co-occurrences of products in baskets). A drawback of this approach is that it takes into account only the last product added to the basket, ignoring the previous ones, which are also valuable for predicting the next product. For example, suppose there are {*milk*, *sugar*, *egg*} in the current shopping cart. Considering all three products is a better indication of buying *flour* rather than considering only *egg*, the last product. Moreover, since both approaches rely on the purchase data, they are poor at modeling products which have few purchases. Particularly, they cannot model new products which have not been purchased by any customers. This problem is known as the *cold-start* problem. As a sequence, it is inevitable to exploit *auxiliary information* such as textual contents in addition to the purchase data (i.e., the interaction data).

Addressing the cold-start using textual contents is well-studied in the literature, particularly in recommender systems (Hu et al., 2008; Li and She, 2017; Wang et

al., 2015a). Generally, these methods are a combination of a text model (e.g., Latent Dirichlet Allocation (LDA) (Wang and Blei, 2011), Stacked Denoising AutoEncoder (Wang et al., 2015a) or a Variational AutoEncoder (Li and She, 2017)) and a matrix factorization (MF)-based model (Salakhutdinov and Mnih, 2008). The key idea behind these approaches is to learn item representations from texts that are useful for predicting the elements of a user-item matrix. Despite being effective approaches, they are not applicable to shopping basket modeling. Indeed, though we can represent the collection of shopping baskets by a *basket-product* matrix, whose rows correspond to baskets and columns correspond to products and factorize this matrix in the same way as the user-item matrix, this is not appropriate for modeling shopping baskets. The reason is that the size of the basket-product matrix changes frequently, every time a basket is recorded, and re-factorizing this matrix frequently is mostly impossible.

Recently, neural network-based approaches archive tremendous success in learning text representations. Such approaches include unsupervised learning approaches ranging from simply composition of the word vectors (Mikolov et al., 2013b; Pennington et al., 2014) to more complicated network architectures such as recurrent neural networks and convolutional neural networks (Chen, 2017; Le and Mikolov, 2014). These approaches outperform traditional text representations including bag-of-words (Manning et al., 2008), Latent Semantic Analysis (LSA) (Deerwester et al., 1990; Hofmann, 1999) or Latent Dirichlet Allocation (LDA) (Blei et al., 2003) in various text understanding tasks. Though they are effective in learning text representations, applying them to learning product representations may not be appropriate for understanding shopping baskets. The reason is that the text representations learned by these models can only capture the semantic similarities of the texts, but cannot capture the relationships between texts that co-occur in baskets. For example, existing text representation learning methods fail to capture that *milk* and *flour* often co-occurs in baskets because there is no semantic similarity between the titles of these products.

This work: To address the aforementioned problems, we propose BASTEXT, a framework for texts and shopping transaction data. In BASTEXT, products are represented by latent factor vectors, the vector representations that are useful for capturing the shopping behaviors of customers. BASTEXT learns such vector representations jointly from the purchase data and the texts associated with the products

(e.g., product titles or product descriptions) by leveraging the recent success of deep learning in natural language processing. BASTEXT is a general framework which can be implemented with various network architectures. In addition, BASTEXT enables to take advantages of pre-trained word vectors such as word2vec (Mikolov et al., 2013b) or GloVe (Pennington et al., 2014) for learning better representations.

We fit BASTEXT into two real-world shopping transaction datasets. We consider the *next product recommendation* task where we predict the products to be added to a shopping basket given the products currently in the task. The experiments show that BASTEXT outperforms the competing methods in this task. We also found that the representations learned by BASTEXT are useful in two real-life scenarios of recommendation: *similar products* recommendation and *also-buy products* recommendation of a given product. Further, we also demonstrate the effectiveness of the learned representations via two tasks: product search via keyword-based query and product category classification. In both tasks, BASKTEXT shows the improvements over the competing methods.

Our main contributions are as follows.

- We propose BASTEXT, a novel end-to-end model for learning product representations from texts which are effective for modeling millions of shopping baskets.
- Our intensive experiments on two real-world datasets demonstrate that BASTEXT significantly outperforms the competing methods in next product recommendation task.
- We also show the effectiveness of the learned representations by performing qualitative analysis, keyword-based product search, and product category classification.

The rest of this section is organized as follows. In section 5.2 we present the BASTEXT model. Section 5.3 presents the experiments and results. Section 5.4 concludes the work and discusses about its issues and future work.

5.2 BASTEXT: The Shopping Basket Model

We propose BASTEXT, a factor model that jointly models the texts associated with products and the shopping basket data. We describe the model and how to learn the

model parameters.

5.2.1 Notations and Definitions

Suppose that we have a collection of T previous shopping baskets. The products in the baskets come from a set of M products, which are denoted by their indices $1, 2, \ldots, M$.

For each product i, there is a text s_i (e.g., *product title*, *product description*, or the set of *tags*) associated with it. We use $\mathbf{w}_i$ to denote the input vector of the i^{th} word. Table 5.1 lists the relevant notations used throughout this section.

Table 5.1: The notations used throughout the section.

Notation	Meaning
T	the number of shopping baskets
M	the number of products
s_i	the text associated with i^{th} product
$\mathbf{S}_B$	the set of texts associated with the products currently in basket B
$\mathbf{w}_i$	the input vector of the i^{th} word in the vocabulary $\mathcal{V}$
d	the embedding size
$\mathbf{h}_i$	the *embedding vector* of product i
$\mathbf{h}'_i$	the *context vector* of product i
$\mathbf{W}$	the connection matrix for a text encoder
$\overline{\mathbf{h}}'_B$	the average of the context vectors of the products currently in basket B
$\mathcal{D}^+$	the set of *positive examples*
$\mathcal{D}^-$	the set of *negative examples*
$\mathcal{D}$	the set of *all examples*: $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$
n	the *negative sampling ratio*

Definition 1 ***(Basket)*** *A basket is a set of products that an individual customer buys in a single shopping trip. A basket B is represented by a set $\{i_1, i_2, \ldots, i_{m_B}\}$ where m_B is the number of products in the basket B, and $\{i_1, i_2, \ldots, i_{m_B}\}$ is a subset of $\{1, 2, \ldots, M\}$.*

Problem 5.1 ***(Next product recommendation)*** *The task is to recommend the next product to add to the current shopping basket given the products already in the cart.*

5.2.2 Next Product Choice

This section presents the decision process of how a customer chooses a product to add to the current basket. We assume the customer identity is anonymous because many websites allow shopping without account registration. We posit that the customer adds products into the basket sequentially. At each step, the customer chooses one product from the available ones, conditioning on the products currently in the basket.

Given current shopping basket B and a potential product i. We are interested in modeling the *probability* that i is added to the basket B.

BASTEXT associates each product i with two latent vectors: *embedding vector* $\mathbf{h}_i$ and *context vector* $\mathbf{h}'_i$ with different roles. The embedding vector $\mathbf{h}_i$ accounts for the attributes of product i. Similar products should have their embedding vectors located close in the latent space. On the other hand, the context vector $\mathbf{h}'_i$ governs the products that appear together with i in the same basket. With these two vectors, we are not only able to model the attributes of a product but also to model the relationships between products via their co-occurrences in shopping baskets.

We define the probability that a potential product i will be added to current basket B as follows.

$$p(next = i|B) = \sigma(\mathbf{h}_i^\top \overline{\mathbf{h}}'_B) \tag{5.1}$$

where $\overline{\mathbf{h}}'_B = \frac{1}{|B|}\sum_{j\in B}\mathbf{h}'_j$ (i.e., the average of the context vectors of the products currently in the basket), and $\sigma(.)$ is the sigmoid function: $\sigma(x) = 1/(1+e^{-x})$.

After learning the *embedding vector* $\mathbf{h}_i$ and *context vector* $\mathbf{h}'_i$ of every product, we can use them for predicting the product to be added in to the current basket. In order to learned the parameters, a training set of *positive examples* and *negative examples* are needed. We will describe how to form such examples in Section 5.2.4.

The embedding and context vectors of products can be learned from only transaction data. However, this way cannot be applied to new products which do not have any purchases. This problem is known as the *cold-start* problem. In order to address the cold-start problem, we are interested in learning these vectors directly from texts.

5.2.3 Dual Text Encoders for Shopping Basket Data

The general architecture is shown in Fig.5.1. Given the texts of products currently in the basket B (on the right side) and the text of a potential product i (on the left side), the model estimates the probability that i will be added to the basket.

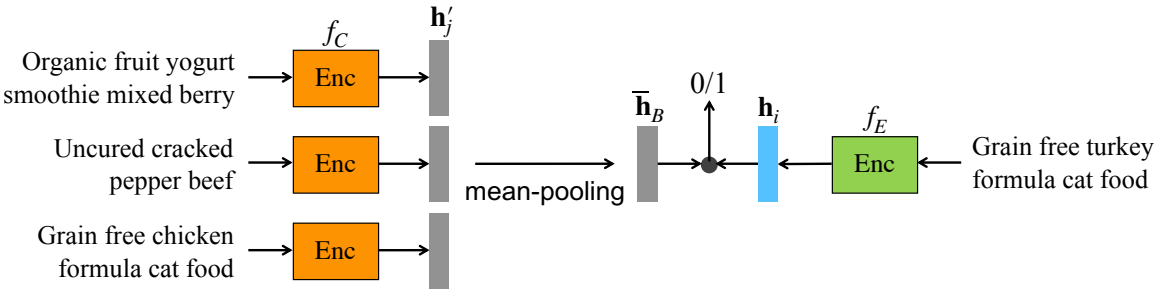


Figure 5.1: The general architecture of BASTEXT framework.

A key component of this model is the *text encoder*, a function f to map a variable-length s text to a fixed-size vector:

$$f(s) \in \mathbb{R}^d \tag{5.2}$$

where d is the embedding size.

Here, we use two text encoders, f_E and f_C, for generating the embedding and context vectors respectively:

$$\begin{aligned} \mathbf{h}_i &= f_E(s_i) \\ \mathbf{h}'_i &= f_C(s_i) \end{aligned} \tag{5.3}$$

Formally, this model is described as follows. For a given basket B and a potential product i, the probability that i will be added in B (Eq.5.1) is equivalent to the probability that the next text is s_i given a set of texts $\mathbf{S}_B$.

$$p(s_i|\mathbf{S}_B) = \sigma\Big(\frac{1}{|B|} f_E^\top(s_i) \sum_{j \in B} f_C(s_j)\Big) \tag{5.4}$$

Here, $f_E(.)$ and $f_C(.)$ are two text encoders which have same architecture but with different weights. We can see that this model extends the concept of the continuous bag of words (CBOW) for word embedding to the texts, where now the role of words is replaced by texts. In training this model, we need a set of positive examples and negative examples. We will show how to form such examples in Section 5.2.4.

The architecture of a text encoder is a modeling choice. It can be simply the average of the vector representations of its words, a convolutional neural network (CNN) or a recurrent neural network (RNN). In the experiments of this work, we implemented two types of text encoders: (1) Mean of vectors (MoV)-based text encoder, and (2) CNN-based text encoder.

MoV-based Text Encoder. This is a simple-but-effective neural network for learning text representation. The representation of a text is simply the mean of the representations of words contained in it.

The input of this network are the input vectors of the words contained in a text. These input vectors can be either *one-hot* vectors of words (end-to-end training) or *pre-trained* vectors obtained from word embedding models such as word2vec (Mikolov et al., 2013b) or GloVe (Pennington et al., 2014). First, these word input vectors are embedded into a low-dimensional vector space via the embedding layer. Then, we use the average of the word embedding vectors as the representation of the text. In order to introduce non-linearity, we added ReLU as an activation function after the average layer.

Formal specification of the MoV-based Text Encoder is as follows.

$$f(s) = ReLU\Big(\frac{1}{|s|}\sum_{l}\mathbf{w}_l^\top\mathbf{W}\Big) \tag{5.5}$$

where $\mathbf{W}$ is the connection matrix of the embedding layer.

Though this network is simple, it has two advantages: (1) it is very efficient in computational cost, (2) it can be used even if there is no explicit order of the words in a text, e.g., when the text is a set of tags associated with a product.

CNN-based Text Encoder. Although the MoV-based Text Encoder is simple and efficient, it ignores the order of words in sentences, which is helpful in learning text representations. Although RNN can retain the order of words, we study CNN due to its efficiency in computational cost over RNN. Here, we adopt the CNN architecture proposed in (Kim, 2014).

5.2.4 Training Data Forming

We present how to form the training data from a collection of shopping baskets.

Each training example is a tuple: $\big\langle(B, i), L\big\rangle$ where B is the set of products cur-

rently in the basket, i is a potential product, and L is the label $+$ (positive example) or $-$ (negative example) to indicate whether actually i was chosen or not.

Forming positive examples. For each basket, we form the positive examples as follows. We pick each product in turn and use as the potential product, the remaining products are used as the products already in the basket. By this way, we obtain m_B positive examples in the form $\langle (B, i), + \rangle$ where m_B is the number of products in the basket B. We use $\mathcal{D}^+$ to denote the set of positive examples.

Forming negative examples. Since in the transaction data, negative examples are not available, we will obtain such examples by sampling from the products that do not appear in the basket. This process is known as *negative sampling*. Strategy for negative sampling is a modelerâĂŹs choice. Here, we adopt the uniform sampling method. Other strategies are left for future work. For each positive example $\langle (B, i), + \rangle$, we randomly pick a product j which is not in the basket to form a negative example $\langle (B, j), - \rangle$. For each positive example, we repeat this procedure n times to obtain n negative examples. We use $\mathcal{D}^-$ to denote the set of negative examples.

5.2.5 Model training

After forming the training data, we have a set of examples $\mathcal{D}$, where each example is in the form $\langle (B, i), L \rangle$, where L is $-$ or $+$.

Our objective function is the negative log likelihood over all examples in the training set. This objective function is formulated as.

$$\mathcal{L}(\Theta) = \sum_{(B,i) \in \mathcal{D}^+} \log \mu_{B,i} - \sum_{(B,i) \in \mathcal{D}^-} (1 - \log \mu_{B,i}) \tag{5.6}$$

where:

$$\mu_{B,i} = \sigma(\mathbf{h}_i^\top \overline{\mathbf{h}}'_B) \tag{5.7}$$

Training the BASTEXT model can be efficiently performed by back-propagation using stochastic gradient descent with mini-batch. We use Adam (Kingma and Ba, 2015), a kind of stochastic gradient descent as the optimizer. Note that we do not perform negative sampling in advanced, which can only produce a fixed set of negative samples. Instead, we use negative sampling at each mini-batch, which let the negative examples change time by time, producing diverse negative examples. The

training process is summarized in Algorithm 4.

Algorithm 4: The training process using back propagation of the BASTEXT model.

Input :

- $\mathcal{D}^+$: the set of positive examples
- n: number of negative samples per positive example

Output: the weights of the network

Initialization: initialize the network's weights using Gaussian distributions
for *epoch=1 ... T* **do**
    Sample negative examples $\mathcal{D}^-$
    $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$
    $\mathcal{O} = \text{Shuffle}(\mathcal{D})$
    for *t=1 ... # of mini-batches* **do**
        $\mathcal{B} = \text{next-mini-batch}(\mathcal{O})$
        $\text{Backprop}(\mathbf{\Theta}, \mathcal{B})$
    end
end

5.3 Experiments

We perform both (1) quantitative experiments: to predict the next products to be added to a shopping basket, and (2) qualitative experiments: to evaluate the effectiveness of the product representations in capturing the relationships between products.

5.3.1 Datasets

In this section, we evaluate the performance of BASTEXT on two public datasets of varying sizes. The datasets are:

- **OnlineRetail** (Chen et al., 2012a): a dataset of online retail transactions provided at the UCI Machine Learning Repository[1]. It contains about 20,059 shopping transactions from December 1, 2010 to December 9, 2011 for a UK-based online retailer. The average number of products in a basket is 26.7. Each

[1] https://archive.ics.uci.edu/ml/datasets/Online+Retail

product is associated with a product description whose average length is 4.3 words.

- **Instacart**[2]: this dataset contains 3.2 million orders, where the average number of products per order is 10.6. Each product is associated with a product name whose average length is 4.7 words.

5.3.2 Experimental Setup

We randomly split the baskets into three set of baskets: *training baskets*, *validation basket* and *testing baskets*, with proportions $85\%, 5\%, 10\%$, respectively. The remaining baskets are used to form the training set. From these set of baskets, we form the *training set*, *validation set* and *test set* in two ways: **warm-start** and **cold-start**. The details about data splitting is given in Table 5.2a and Table 5.2b.

Table 5.2: The statistical information of the datasets

(a) Warm-start splitting

Data	OnlineRetail	Instacart
# training baskets	17K	2.7M
# validation baskets	1K	159K
# test baskets	1.9K	318K
# test cases	51K	3.3M

(b) Cold-start splitting

Data	OnlineRetail	Instacart
# training baskets	16K	2.3M
# validation baskets	988	138K
# test baskets	1.7K	312K
# test cases	13.6K	2.3M

Warm-start. In this setting, we make sure that every product in the test set appears in the training set. To do that, we remove from the test baskets the products that do not appear in the training baskets. The test cases are constructed as follows: in each testing basket, we repeatedly pick one product as the ground-truth potential product, the remaining products are used as the products already in the shopping

[2]"The Instacart Online Grocery Shopping Dataset 2017", Accessed from https://www.instacart.com/datasets/grocery-shopping-2017

basket. We form the validation set using the validation baskets, following the same way as we do with the test set. The remaining baskets from the training set (after picking the validation set) are used to form the training data as described in Section 5.2.4.

Cold-start. In this setting, we make sure that every product in the test set is absent from the training baskets. We randomly pick 10% products from the test baskets and call these products *test products*. We remove these products from train baskets (to make sure the test products do not appear in training baskets). Then we form the test cases in which the potential products come from the *test products*. The validation set and training set are formed similarly with the warm-start setting.

Evaluation Metrics For each basket in the test set, we predict the relevant scores for all the remaining products and rank these products according to their relevance scores. We then pick N products that have highest scores to form a recommendation list (top-N recommendation). An accurate recommendation list should contain the products in the held-out set.

We use rank-based metrics: Recall@N, and MRR@N (mean reciprocal rank) for evaluating the models. Recall@N measures the proportion of cases out of all test cases in which the held-out item is listed in the top-N products. The MRR@N considers the rank of the relevant product within the top-N to be equivalent. Recall@N and MRR@N are formulated as follows.

$$\text{Recall@}N = \frac{1}{|V|}\sum_{i=1}^{V}\mathbb{I}\{h_i \in S_i(N)\} \tag{5.8}$$

where $|V|$ is the number of test cases in the test set, $S_i(K)$ is the set of top-N products, h_i is the held-out product of test case i, and $\mathbb{I}\{.\}$ is the indicator function.

$$MRR@N = \frac{1}{V}\sum_{i=1}^{V}\frac{1}{rank_i} \tag{5.9}$$

where $rank_i$ is the rank of the held-out product in the top-N products. The reciprocal rank is set to zero if the rank is larger than N.

Competing Methods. In evaluating the predictive performance, we compare the following methods (including ours).

- **POP** (popular products): this model recommends the most popular products in the training set. Though this model is simple, POP is often a strong baseline

in certain domains.

- **ItemKNN** (Linden et al., 2003): this model provides a product recommendation based on its co-occurrence with other products along baskets. This approach is one of the most common *item-to-item* recommendation, which is in the form "consumers who bought X also bought Y".

- **prod2vec** (Grbovic et al., 2015): a word2vec (Mikolov et al., 2013b) version for learning the representations products by corresponding each basket as a sentence and each product in the basket as a word. The representation of a basket is calculated as the mean of the representations of the products contained in the basket. Given a basket, we compute *cosine similarities* between its representation and all available products, and pick top N-similar products.

- **doc2vec** (Le and Mikolov, 2014): a model for learning representations of texts. We run doc2vec on the set of product titles to obtain the representation of the products. The representation of a basket is calculated as the mean of the representations of products contained in it. Given a basket, we calculate the *cosine similarities* between the representation of the basket and all potential products, and pick top-N similar products.

- **BASTEXT-Avg** (our): the BASTEXT model where the MoV-based text encoders are used for learning text representations. The word input vectors are one-hot vectors.

- **BASTEXT-Avg+w2v** (our): the BASTEXT-Avg model where the input vector for each word is the pre-trained word vector (Pennington et al., 2014).

- **BASTEXT-Conv** (our): the BASTEXT model where the CNN-based text encoder is used for learning the representations of texts. The input vector for each word is its one-hot-vector.

- **BASTEXT-Conv+w2v** (our): the BASKET-Conv model where the input vector for each word is the pre-trained word2vec vector (Pennington et al., 2014).

5.3.3 Implementation Detail

We implemented the proposed model based on Pytorch. All BASTEXT variants are trained by optimizing the binary cross-entropy loss in Eq.(5.6) where the embedding size is 64 and the negative sampling ratio is 8 for each positive examples. We use dropout (Srivastava et al., 2014) with dropout rate 0.3 for hidden layers to avoid over-fitting. All the layers of the network are initialized using Gaussian distribution with $mean = 0$ and standard deviation $std = 0.01$. Further, we vary the embedding size d and the negative sampling ratio n for studying the impact of the embedding size and the negative sampling ration on the accuracy of the proposed model. The learning rate used in the experiments is $lr = 0.01$, the weight decay is 0.01.

To speed up the training process, we exploit the power of GPU. In dividing the training data into mini-batch, we choose the mini-batch size that fit the GPU's memory and performs the gradient descent for each mini-batch on GPU. We use mini-batch size 10,000 for OnlineRetail dataset and 5,000 for Instacart dataset. We use early stopping to terminate the training process if the loss function does not decrease on the validation set for five epochs.

5.3.4 Predictive Performance Comparison

In this session, we compare the performances of next product recommendation of the models.

Comparison over baselines. Table 5.3 and Table 5.4 show the performances of next product prediction task of the methods in both warm-start and cold-start settings. We can see that all variants of BASTEXT significantly outperform the competing methods on all datasets. Besides, we have following observations.

As expected, POP does not achieve good performance because it cannot capture the context of the shopping trips. It is easily beaten by ItemKNN and prod2vec which can capture the relationships between products. prod2vec is slightly better than ItemKNN, suggesting that prod2vec can capture the relationships between products better than ItemKNN.

In both datasets, we can see that ItemKNN and prod2vec outperform doc2vec, which uses content only. The gap between doc2vec and prod2vec and ItemKNN indicates that product relationships are more valuable than the contents in capturing the shopping behaviors.

Table 5.3: Recall and MRR for next product recommendation (warm-start setting). Here, we fixed the embedding size $d = 64$ and negative sampling ratio $n = 8$.

(a) OnlineRetail data

Methods	Recall@10	Recall@20	MRR@20
POP	0.0828	0.1212	0.0652
ItemKNN	0.1923	0.2511	0.1765
prod2vec	0.2007	0.2632	0.1876
doc2vec	0.1773	0.2332	0.1521
BASTEXT-Avg (our)	**0.2181**	**0.2854**	**0.1943**
BASTEXT-Avg+w2v (our)	**0.2275**	**0.2942**	**0.2132**
BASTEXT-Conv (our)	**0.2212**	**0.2897**	**0.1998**
BASTEXT-Conv+w2v (our)	**0.2378**	**0.3096**	**0.2251**

(b) Instacart data

Methods	Recall@10	Recall@20	MRR@20
POP	0.0124	0.0153	0.0102
ItemKNN	0.1065	0.1507	0.0985
prod2vec	0.1251	0.1623	0.1048
doc2vec	0.0912	0.1215	0.0981
BASTEXT-Avg (our)	**0.1527**	**0.1932**	**0.1329**
BASTEXT-Avg+w2v (our)	**0.1631**	**0.2013**	**0.1521**
BASTEXT-Conv (our)	**0.1578**	**0.1965**	**0.1401**
BASTEXT-Conv+w2v (our)	**0.1698**	**0.2102**	**0.1598**

In both datasets, BASTEXT-Avg outperforms both prod2vec. It indicates that, although prod2vec works well for the warm-start setting, introducing textual contents will significantly improve the predictive performances. The improvements of BASTEXT-Avg over prod2vec are 8.4% and 19% for OnlineRetail and Instacart, respectively.

In the cold-start setting, only doc2vec and the variants of BASTEXT work. The performances of doc2vec are almost the same as the warm-start setting because it uses content only. Although BASTEXT-Avg's performances decrease, compared with the warm-start setting, it still performs slightly better than doc2vec. This indicates that jointly training the texts with purchase data will improve the performance.

Impact of the Text Encoder model. From Table 5.3 and 5.4 we can observe that BASTEXT-Conv and BASTEXT-Conv+w2v slightly perform better than their

Table 5.4: Recall and MRR for next product prediction (cold-start setting). Here, we fixed the embedding size $d = 64$ and negative sampling ratio $n = 8$.

(a) OnlineRetail data

Methods	Recall@10	Recall@20	MRR@20
doc2vec	0.1768	0.2315	0.1532
BASTEXT-Avg (our)	**0.1823**	**0.2378**	**0.1628**
BASTEXT-Avg+w2v (our)	**0.1861**	**0.2432**	**0.1679**
BASTEXT-Conv (our)	**0.1842**	**0.2397**	**0.1642**
BASTEXT-Conv+w2v (our)	**0.1908**	**0.2483**	**0.1733**

(b) Instacart data

Methods	Recall@10	Recall@20	MRR@20
doc2vec	0.0916	0.1208	0.0977
BASTEXT-Avg (our)	**0.1021**	**0.1297**	**0.1048**
BASTEXT-Avg+w2v (our)	**0.1098**	**0.1385**	**0.1195**
BASTEXT-Conv (our)	**0.1075**	**0.1342**	**0.1127**
BASTEXT-Conv+w2v (our)	**0.1127**	**0.1428**	**0.1249**

counterparts (BASTEXT-Avg and BASTEXT-Avg+w2v). This indicates that taking into account the order of words in texts can improve the effectiveness of the representations. However, such minor improvements suggest that, for short texts, using the text representations as the average of the representations of its words is also a strong baseline, given that its complexity is much cheaper than a convolutional neural network.

Impact of pre-trained word input vectors. One advantage of BASTEXT is that it can use the pre-trained word embedding vectors such as word2vec (Mikolov et al., 2013b) or Glove (Pennington et al., 2014) as word input vectors. Thus, we can study the impact of using pre-trained word embedding vectors on the performance of BASTEXT. Here, we adopt the pre-trained vectors of Glove (Pennington et al., 2014).

Table 5.3 and Table 5.4 show marginal improvements of BASTEXT-Avg+w2v over BASTEXT-Avg and BASTEXT-Conv+w2v over BASTEXT-Conv for both datasets. It means that using the pre-trained word vectors significantly improves the performance of the model. It is because the product titles are very short, and therefore, are poor at capturing the semantic meaning of the texts. Using pre-trained word vectors

will help improve the representation effectiveness of short texts.

5.3.5 Product-based Recommendation

Product-based recommendation, i.e., making recommendations in the context of a specific product, is a typical setting in recommendation. Here, we consider two kinds of such recommendations: *similar products* recommendations and *also-buy products* recommendations.

Similar product recommendation. This kind of recommendation is very useful in real-life, particularly, when a customer is considering a product to buy. For example, if the customer is considering (i.e., viewing) a $skirt$, it makes sense to show her some other skirts so that she can compare before deciding.

Given two products i and j, the similarity between these products is defined as the *cosine similarity* between their embedding vectors. This definition of similarity is natural due to the assumption of the model: the embedding $\mathbf{h}_i$ reflects the attributes of products i, thus, products with similar attributes should have their embedding vectors located closed in the latent space. The similarity between two products i and j is as follows.

$$sim(i, j) = cosine(\mathbf{h}_i, \mathbf{h}_j) \tag{5.10}$$

Figure 5.2: Similar product recommendation. For each row, the product in the left side is a "query" product, following by its top-3 similar products.

Fig.5.2 shows some examples of *similar products* recommendations made by BASTEXT. In this figure, for each row, the most left is a "query" product, and the next three products are top-3 products that are frequently purchased together with it, calculated by Eq.(5.10).

Figure 5.3: Also-buy product recommendation. For each row, the product in the left side is the "query" product, following top-3 products that are often co-purchased with it.

Also-buy Product Recommendation. This is to recommend products that are frequently purchased together with a specific product (e.g., of the product that the customer is viewing, or of the product that the customer has just added to the basket). This scenario is useful, particularly, when a customer added a product to his shopping basket.

Given two products i and j, we compute how likely i is bought given that the customer has already bought j as the *inner product* of the context vector $\mathbf{h}_i$ and $\mathbf{h}'_j$:

$$Also\_buy(i, j) = \mathbf{h}_i^\top \mathbf{h}'_j \tag{5.11}$$

Fig.5.3 shows some examples of *Also-buy products* recommendations made by BASTEXT. In this figure, for each row, the most left is a "query" product, and the next three products are top-3 products that are frequently purchased together with it, calculated by Eq.(5.11).

5.3.6 Effectiveness of the Representations

For understanding more about the insights of BASTEXT, we study how well BASTEXT represents the texts and how well it captures the semantics behind the products. To do so, we perform two tasks: *keyword-based product search* and *product category classification*.

Keyword-based product search. We are provided a query s in the form of keywords, the task is to retrieve the products relevant to the query. We compare

BASTEXT with doc2vec (Le and Mikolov, 2014), a sentence representation learning model.

First, we infer the vector representations of the query by two models BASTEXT-Avg and doc2vec. For BASTEXT, we use the text encoder f_E. We then compute the cosine similarity of the vector representations of these two models with the embedding vector of every product in the dataset. Top-5 similar products are reported in Table 5.5.

We can observe that BASTEXT retrieves more relevant products than doc2vec does. Particularly for the second query, *natural herb cough drops*, BASTEXT, generally, captures the meaning of the query and can return relevant products. On the other hand, doc2vec completely misunderstands the query. We found that the keywords of this query rarely appear in the corpus, therefore, doc2vec cannot learn good representations of them. BASTEXT, in contrast, utilizing the purchase data for capturing effective representations of texts. This experiment suggests that BASTEXT can be a potential baseline for product search via keywords, particularly when each product is associated with a very short text.

Category classification. We additionally study the effectiveness of the product representations of BASTEXT, prod2Vec, and doc2vec, by performing category classification task on Instacart dataset. To do so, we use the embedding vectors of the products learned by these models as the feature vectors for training classifiers. We use SVM (Support Vector Machine) as the classifier. We perform 5-fold cross-validation and report the classification accuracies. The products used in the test are from two groups. The first group contains top-5 most active categories (categories that are most frequently purchased): (H1) Produce, (H2) Dairy eggs, (H3) Snacks, (H4) Beverages, (H5) Frozen. The second group contains top-5 *less active* categories (categories that are less frequently purchased): (L1) Personal care, (L2) Babies, (L3). (L4) Alcohol, (L5) Pets.

The category classification accuracy is shown in Fig.5.4. We have following observations. First, the accuracies of doc2vec are almost the same across the categories. This is because doc2vec uses only the textual content, ignoring the purchase data. In the *less active* categories (L1-L5), BASTEXT and prod2vec perform better than doc2vec, however, the differences are not big. The differences between BASTEXT and prod2vec with doc2vec become larger in the *most active* categories (H1-H5), indicating the important role of purchase data in the performance. In the *most active*

Table 5.5: Keyword-based product search results performed on Instacart dataset. The top line are the query (boldface font). Below the query are top five answers by BASTEXT-Avg and word2vec, respectively. Inside the braces () are the categories of the returned products. Underlined words are words appearing in the query.

query	**organic tea**
BASTEXT	organic honeybush tea (tea) organic chamomile lemon tea (tea) organic oolong tea bags (tea) organic white rose white tea (tea) chinese breakfast black tea (tea)
doc2vec	organic english breakfast black tea (tea) lemon sweet tea iced tea mix (tea) organic grapefruit honey lightly sweetened iced green tea (tea) bags organic turmeric ginger green tea (tea) half sweet tea pink lemonade (tea)
query	**natural herb cough drops**
BASTEXT	cough drop (cold flu allergy) honey/lemon cough drops (cold flu allergy) immune+ super orange drink mix dietary supplement (cold flu allergy) defense vitamin c, cold flu allergy (cold flu allergy) natural throat drops honey & pomegranate (cold flu allergy)
doc2vec	ultra thin crust cheese lovers pizza (frozen pizza) homemade pizza sauce (pasta sauce) deep dish sausage pizza singles (frozen pizza) authentic deep dish sausage pizza (frozen pizza) colby jack cheese (packaged cheese)

categories, BASTEXT is still better than prod2vec. This implies that introducing the associated texts will improve the effectiveness of the representations.

5.3.7 Hyper-parameter Sensitivity

In this section, we study the impact of hyper-parameters including the *negative sampling ratio* n, and the *embedding size* d to the performance of the next product prediction.

Impact of the Negative Sampling Ratio. Fig.5.5 shows the next product prediction performances of BASTEXT-Avg (other variants of BASTEXT have similar behaviors). We observe that the trends of the performances are quite similar. For both

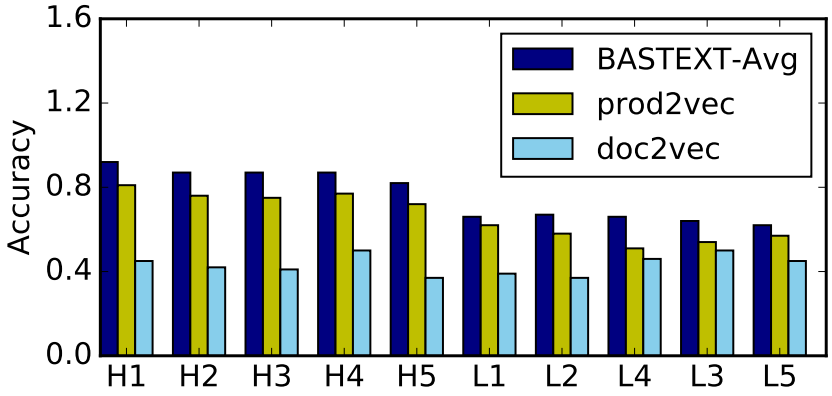
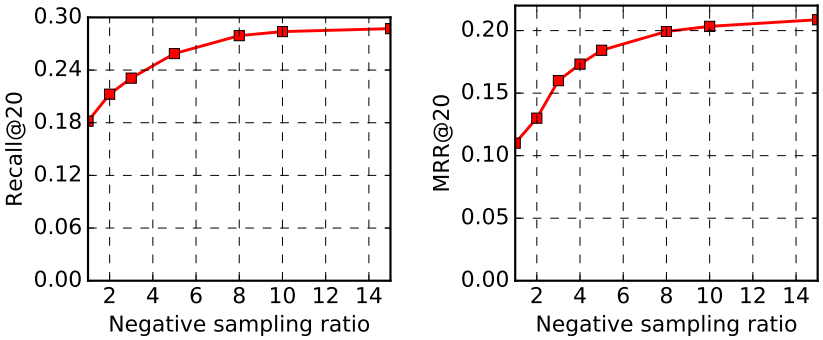


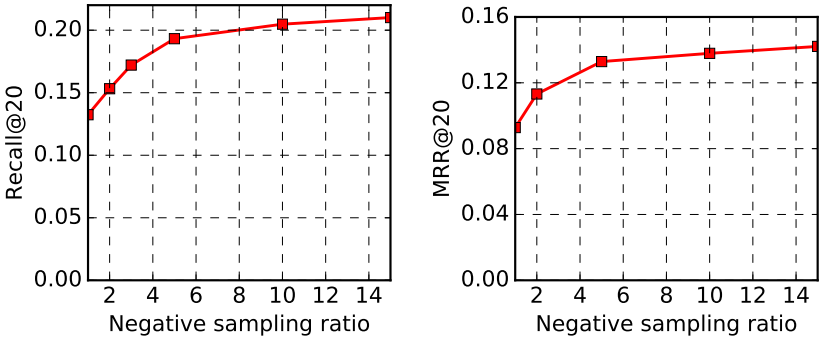
Figure 5.4: Performance of product category classification on Instacart.

datasets, the Recall@20 and MRR@20 increase when the negative ratio n increases, until a certain value of n (around 8 to 10). From this threshold, the performances are quite stable (do not increase much). Therefore, we do not need to sample more negative examples than this threshold.

Impact of the Embedding Size. We study the effects of the embedding sizes d to the performance of BASTEXT. Results are shown in Fig.5.6. Here, we report the results of BASTEXT-Avg model only, because, other variants of BASTEXT have similar behaviors. From the results, we can observe that the performances increase when d increase, until a threshold of d. After this threshold, the performances decrease due to overfitting or do not significantly increase. For OnlineRetail dataset, the performances decrease when $d > 64$. On the other hand, for Instacart, the performances continue increasing after $d > 64$, however, the improvement is not significant. These observations suggest that the embedding size around $d = 64$ will have a balance between the performances and the computational complexity.

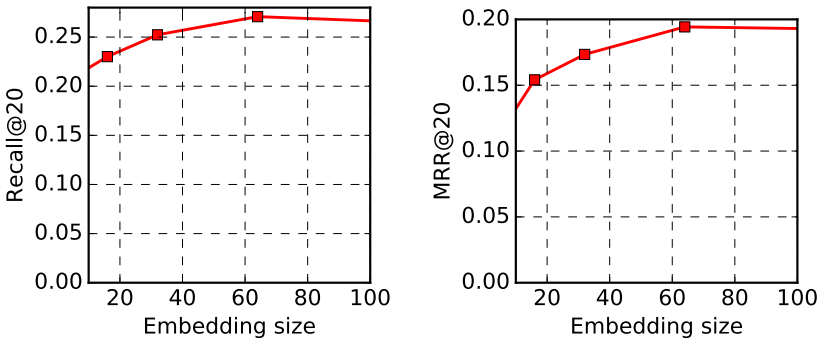


(a) Re@20 and MRR@20 for OnlineRetail.

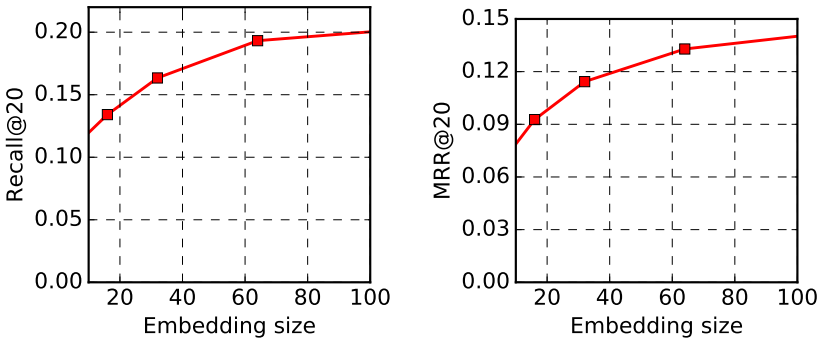


(b) Re@20 and MRR@20 for Instacart.

Figure 5.5: Impact of the negative sampling ratio. Here we use BASTEXT-Avg with embedding size $d = 64$.



(a) Re@20 and MRR@20 for OnlineRetail.



(b) Re@20 and MRR@20 for Instacart.

Figure 5.6: Impact of the embedding size to the next product recommendation. Here we use BASTEXT-Avg with negative sampling ratio $n = 8$.

5.4 Chapter Summary

We introduced BASTEXT, a joint model of texts and shopping transaction data for modeling the relationships between products. We proposed a dual Text Encoder-based model learning the representations of texts associated with products, that capture the behaviors of customers in shopping. BASTEXT can utilize the texts for addressing the cold-start problem and utilize the purchase data for improving the performance of text representations. Experiments on two real-world datasets demonstrate that BASTEXT outperforms competing methods in next product recommendation. Further, the qualitative analysis demonstrates the effectiveness of the representations learned by the BASTEXT. The success of BASTEXT suggests that it can be a potential multi-purpose recommender system.

Chapter 6

Conclusion

6.1 Contribution Summary

In this dissertation, we propose models for learning the representations of users and items for recommender systems that are effective for the sparse datasets and the cold-start problem.

Implicit Feedback Embedding for Rating Prediction. We develop a model for the rating prediction, an essential task in understanding user preferences. Usually, the rating prediction is suffered from the sparsity of the rating matrix. Furthermore, the items that have no prior ratings can not be predicted (the cold-start problem). In oder to address these issues, we exploit the implicit feedback (e.g., clicks, page views), which is abundant. To do so, we propose a probabilistic item embedding which learn the embedding vectors of the items from the implicit feedback. The embedding is performed by factorizing the PPMI matrix, which is constructed from the implicit data. We then propose CoMF, a joint model of the item embedding model and the rating prediction model. CoMF simultaneously factorizes the PPMI matrix and the rating matrix in a shared latent space. In this method, the features of items are extracted based on two sources of feedback: rating data and click data. We show that exploiting the click data can supplement the shortage of rating data and is able to deal with the cold-start issue.

Neural Personalized Embedding Model. We address a fundamental cold-start issue of the collaborative filtering: it cannot recommend for the new users who have only very few clicks. We proposed a neural personalized embedding model for modeling the clicks of users to items. In this model, we assume that a click of a user to

an item is governed by two signals: the preferences of the user to this item and the compatibility between this item and the items that the user previously clicked. The experimental results on real-world datasets show the improvements over competing methods. Further, the proposed model is also able to detect the similar items and the "purchased together" items of a given item.

Learning Representations from Product Titles for Shopping Basket Collection Modeling. We introduce a model to learn the representations of the products based on their titles to model a collection of shopping transactions. The learned representations can help identify two kinds of relationships between products: the *similar products* and *also-buy products*. The proposed model can be used for multiple purposes: next product recommendation, similar product recommendation, also-buy product recommendation, and product search by keywords. Because the model learns the representations from product titles, it can deal with the cold-start problem, the problem of modeling new products which have not appeared in any shopping transactions.

6.2 Future Work

We have some directions for extending the current research as follows.

Exploit the rich content of the items for fully addressing the cold-start problem. Addressing the cold-start problem is one essential task of modern recommender systems. In Chapter 3 we proposed CoMF, which is a combination of the matrix factorization for rating data and the item embedding from the click data. CoMF can predict the ratings of the items which do not have any prior ratings as long as they have some clicks. However, in the case an item has no click data, CoMF fails to predict the rating for it. In order to address this problem, the common way is to exploit the rich contents of the items such as textual contents (Li and She, 2017; Wang and Blei, 2011; Wang et al., 2015a) or visual contents (He and McAuley, 2016). These methods are a combination of content models (e.g., LDA or neural networks) and a matrix factorization for rating data. The goal of these methods is to learning the item representations, from the rich contents, that can explain the observed rating data. Therefore, these methods also suffer from the sparsity of the rating matrix. To address this issue, the proposed CoMF can be integrated with content models to leverage the advantage of using click data.

In the NPE model (Chapter 4), the input vectors of the items can be either one-hot-vectors or the feature vectors obtained from the rich contents. In this thesis, we use one one-hot-vectors as the input vectors of the items. As a future work, using feature vectors from the rich contents of the items can fully address the cold-start issue. The rich contents can be integrated into NPE in an end-to-end model or separately obtain the feature vectors using content models such as tf-idf, LDA or neural network and then feed these feature vectors as the input of the NPE model.

Session-based recommendation with multiple feedbacks. In Chapter 5, we proposed BASTEXT, a model for learning the representations of the items from a collection of shopping baskets. In other words, these representations are learned from the purchase data. However, in a real-life system, besides the purchase data we also can collect other kinds of data. For example, a shopper usually views multiple products for examining before decide to add a product into the shopping cart. Such product view data is much more abundant than purchase data and also contain valuable information about the relationships between the products. A model that can model such multiple related feedbacks can help learn high-quality product representations and improve the accuracy of the product recommendations.

References

[Agarwal et al. 1994] Rakesh Agarwal, Ramakrishnan Srikant et al.: Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Databases* (VLDB '94), pages 487–499, September 1994.

[Barkan and Koenigstein 2016] Oren Barkan, and Noam Koenigstein: ITEM2VEC: Neural item embedding for collaborative filtering. In: *Proceedings of the IEEE 26th International Workshop on Machine Learning for Signal Processing* (MLSP '16), pages 1–6, September 2016.

[Bell and Koren 2007] Robert M. Bell, and Yehuda Koren: Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In: *Proceedings of the 7th IEEE International Conference on Data Mining* (ICDM '07), pages 43–52, October 2007.

[Blei et al. 2003] David M. Blei, Andrew Y. Ng, and Michael I. Jordan: Latent Dirichlet Allocation: In: *The Journal of Machine Learning Research*, Vol. 3, pages 993–1022, March 2003.

[Bullinaria and Levy 2007] John Bullinaria, and Joseph Levy: Extracting semantic representations from word co-occurrence statistics: A computational study: In: *Behavior Research Methods*, Vol. 39, No. 3, pages 510–526, August 2007.

[Chen et al. 2012a] Daqing Chen, Sai Laing Sain, and Kun Guo: Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining: In: *Journal of Database Marketing & Customer Strategy Management*, Vol. 19, No. 3, pages 197–208, September 2012.

[Chen 2017] Minmin Chen: Efficient Vector Representation for Documents Through Corruption (ICLR '17), pages 1–13, April 2017.

[Chen et al. 2012b] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims: Playlist Prediction via Metric Embedding. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '12), pages 714–722, August 2012.

[Church and Hanks 1990] K. W. Church, and P. Hanks: Word Association Norms, Mutual Information, and Lexicography: In: *Computational Linguistics*, Vol. 1, No. 16, pages 22–29, March 1990.

[Das et al. 2007] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram: Google News Personalization: Scalable Online Collaborative Filtering. In: *Proceedings of the 16th International Conference on World Wide Web* (WWW '07), pages 271–280, May 2007.

[Deerwester et al. 1990] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman: Indexing by latent semantic analysis: In: *Journal of the American Society for Information Science*, Vol. 41, No. 6, pages 391–407, September 1990.

[Dziugaite and Roy 2015] Gintare Karolina Dziugaite, and Daniel M. Roy: Neural Network Matrix Factorization: In: *CoRR*, Vol. abs/1511.06443, December 2015.

[Gopalan et al. 2014] Prem Gopalan, Laurent Charlin, and David M. Blei: Content-based recommendations with Poisson factorization. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems* (NIPS '14), pages 3176–3184, December 2014.

[Grbovic et al. 2015] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp: E-commerce in Your Inbox: Product Recommendations at Scale. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '15), pages 1809–1818, August 2015.

[He and McAuley 2016] Ruining He, and Julian McAuley: VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence* (AAAI '16), pages 144–150, February 2016.

[He et al. 2017] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua: Neural Collaborative Filtering. In: *Proceedings of the 26th International Conference on World Wide Web* (WWW '17), pages 173–182, April 2017.

[Hidasi et al. 2016] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk: Session-based Recommendations with Recurrent Neural Networks. In: *Proceedings of the 4th International Conference on Learning Representations* (ICLR '16), pages 1–10, May 2016.

[Hofmann 1999] Thomas Hofmann: Probabilistic Latent Semantic Indexing. In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '99'), pages 50–57, August 1999.

[Hu et al. 2008] Yifan Hu, Yehuda Koren, and Chris Volinsky: Collaborative filtering for implicit feedback datasets. In: *Proceedings of the 8th IEEE International Conference on Data Mining* (ICDM '08), pages 263–272, December 2008.

[Kim 2014] Yoon Kim: Convolutional Neural Networks for Sentence Classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (EMNLP '14), pages 1746–1751, October 2014.

[Kingma and Ba 2015] Diederik P. Kingma, and Jimmy Ba: Adam: A Method for Stochastic Optimization. In: *Proceedings of the 3rd International Conference for Learning Representations* (ICLR '15), pages 1–15, May 2015.

[Kiros et al. 2015] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler: Skip-thought Vectors. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems* (NIPS '15), pages 3294–3302, December 2015.

[Koren 2008] Y. Koren: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge discovery and Data Mining* (KDD '08), pages 426–434, August 2008.

[Koren 2009] Yehuda Koren: Collaborative Filtering with Temporal Dynamics. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '09), pages 447–456, June/July 2009.

[Le and Mikolov 2014] Quoc Le, and Tomas Mikolov: Distributed Representations of Sentences and Documents. In: *Proceedings of the 31st International*

Conference on International Conference on Machine Learning (ICML '14), pages 1188–1196, June 2014.

[Lee and Seung 2000] Daniel D. Lee, and H. Sebastian Seung: Algorithms for Non-negative Matrix Factorization. In: *Proceedings of the 13th International Conference on Neural Information Processing Systems* (NIPS '00), pages 556–562, November 2000.

[Levy and Goldberg 2014] Omer Levy, and Yoav Goldberg: Neural Word Embedding as Implicit Matrix Factorization. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems* (NIPS '14), pages 2177–2185, December 2014.

[Li et al. 2015a] Shaohua Li, Jun Zhu, and Chunyan Miao: A Generative Word Embedding Model and its Low Rank Positive Semidefinite Solution. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (EMNLP '15), pages 1599–1609, September 2015.

[Li et al. 2015b] Sheng Li, Jaya Kawale, and Yun Fu: Deep Collaborative Filtering via Marginalized Denoising Auto-encoder. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (CIKM '15), pages 811–820, October 2015.

[Li and She 2017] Xiaopeng Li, and James She: Collaborative Variational Autoencoder for Recommender Systems. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '17), pages 305–314, August 2017.

[Liang et al. 2016] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M. Blei: Factorization Meets the Item Embedding: Regularizing Matrix Factorization with Item Co-occurrence. In: *Proceedings of the 10th ACM Conference on Recommender Systems* (RecSys '16), pages 59–66, September 2016.

[Linden et al. 2003] Greg Linden, Brent Smith, and Jeremy York: Amazon.Com Recommendations: Item-to-Item Collaborative Filtering: In: *IEEE Internet Computing*, Vol. 7, No. 1, pages 76–80, January 2003.

[Liu et al. 2010] Nathan N. Liu, Evan W. Xiang, Min Zhao, and Qiang Yang: Unifying explicit and implicit feedback for collaborative filtering. In: *Proceedings of the 19th ACM international conference on Information and Knowledge Management* (CIKM '10), pages 1445–1448, October 2010.

[Manning et al. 2008] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze: *Introduction to Information Retrieval*. 2008.

[Mikolov et al. 2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean: Efficient estimation of word representations in vector space. In: *Proceedings of the 1st International Conference on Learning Representations* (ICLR '13), pages 1–12, May 2013.

[Mikolov et al. 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean: Distributed Representations of Words and Phrases and Their Compositionality. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems* (NIPS '13), pages 3111–3119, December 2013.

[Nguyen et al. 2017] ThaiBinh Nguyen, Kenro Aihara, and Atsuhiro Takasu: Collaborative Item Embedding Model for Implicit Feedback Data. In: *Proceedings of the 17th International Conference on Web Engineering* (ICWE '17), pages 336–348, June 2017.

[Nguyen and Takasu 2017] ThaiBinh Nguyen, and Atsuhiro Takasu: A Probabilistic Model for the Cold-Start Problem in Rating Prediction Using Click Data. In: *Proceeding of the 24th International Conference on Neural Information Processing* (ICONIP '17), pages 196–205, November 2017.

[Ning et al. 2015] Xia Ning, Christian Desrosiers, and George Karypis: A Comprehensive Survey of Neighborhood-Based Recommendation Methods. In: *Recommender Systems Handbook*, pages 37–76, 2015.

[Ning and Karypis 2011] Xia Ning, and George Karypis: SLIM: Sparse Linear Methods for Top-N Recommender Systems. In: *Proceedings of the 2011 IEEE 11th International Conference on Data Mining* (ICDM '11), pages 497–506, December 2011.

[Oord et al. 2013] Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen: Deep Content-based Music Recommendation. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems* (NIPS '13), pages 2643–2651, December 2013.

[Pan et al. 2008] Rong Pan, Yunhong Zhou, Bin Cao, Nathan Nan Liu, Rajan M. Lukose, Martin Scholz, and Qiang Yang: One-Class Collaborative Filtering. In: *Proceedings of the 8th IEEE International Conference on Data Mining* (ICDM '08), pages 502–511, December 2008.

[Pazzani and Billsus 2007] Michael J. Pazzani, and Daniel Billsus: Content-Based Recommendation Systems. In: *The Adaptive Web: Methods and Strategies of Web Personalization*, pages 325–341, 2007.

[Pennington et al. 2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (EMNLP '14), pages 1532–1543, October 2014.

[Rendle et al. 2009] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme: BPR: Bayesian Personalized Ranking from Implicit Feedback. In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence* (UAI '09), pages 452–461, June 2009.

[Rendle et al. 2010] 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* (WWW '10), pages 811–820, April 2010.

[Rudolph et al. 2016] Maja Rudolph, Francisco Ruiz, Stephan Mandt, and David Blei: Exponential Family Embeddings. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems* (NIPS '16), pages 478–486, December 2016.

[Salakhutdinov and Mnih 2008] Ruslan Salakhutdinov, and Andriy Mnih: Probabilistic Matrix Factorization. In: *Proceedings of the 20th International Conference on Neural Information Processing Systems* (NIPS '07), pages 1257–1264, December 2008.

[Sarwar et al. 2001] 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* (WWW '01), pages 285–295, May 2001.

[Shani et al. 2005] Guy Shani, David Heckerman, and Ronen I. Brafman: An MDP-Based Recommender System: In: *Journal of Machine Learning Research*, Vol. 6, pages 1265–1295, December 2005.

[Singh and Gordon 2008] Ajit Paul Singh, and Geoffrey J. Gordon: Relational learning via collective matrix factorization. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '08), pages 650–658, August 2008.

[Srivastava et al. 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov: Dropout: A Simple Way to Prevent Neural Networks from Overfitting: In: *The Journal of Machine Learning Research (JMLR)*, Vol. 15, No. 1, pages 1929–1958, January 2014.

[Tai et al. 2015] Kai Sheng Tai, Richard Socher, and Christopher D. Manning: Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (ACL-IJCNLP 2015), pages 1556–1566, July 2015.

[Tang and Liu 2017] Lijun Tang, and Eric Yi Liu: Joint User-Entity Representation Learning for Event Recommendation in Social Network. In: *Proceeding of the 33rd IEEE International Conference on Data Engineering* (ICDE '17), pages 271–280, April 2017.

[Wang et al. 2012] Bin Wang, Mohammadreza Rahimi, Dequan Zhou, and Xin Wang: Expectation-Maximization Collaborative Filtering with Explicit and Implicit Feedback. In: *Proceedings of the 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (PAKDD '12), pages 604–616, May/June 2012.

[Wang and Blei 2011] Chong Wang, and David M. Blei: Collaborative Topic Modeling for Recommending Scientific Articles. In: *Proceedings of the 17th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11), pages 448–456, August 2011.

[Wang et al. 2015a] Hao Wang, Naiyan Wang, and Dit-Yan Yeung: Collaborative Deep Learning for Recommender Systems. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '15), pages 1235–1244, August 2015.

[Wang et al. 2015b] 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* (SIGIR '15), pages 403–412, August 2015.

[Weston et al. 2014] Jason Weston, Sumit Chopra, and Keith Adams: TagSpace: Semantic Embeddings from Hashtags. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (EMNLP '14), pages 1822–1827, October 2014.

[Yu et al. 2016] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan: A Dynamic Recurrent Model for Next Basket Recommendation. In: *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '16), pages 729–732, August 2016.