Modeling Rating Relation Vectors in User-Item Interaction for Hybrid Collaborative Filtering

by

Phannakan TENGKIATTRAKUL

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2022

Committee

Advisor	Dr. Atsuhiro TAKASU Professor of National Institute of Informatics/SOKENDAI
Examiner	Dr. Keizo OYAMA Professor of National Institute of Informatics/SOKENDAI
Examiner	Dr. Akiko AIZAWA Professor of National Institute of Informatics/SOKENDAI
Examiner	Dr. Kenro AIHARA Professor of Tokyo Metropolitan University
Examiner	Dr. Saranya MANEEROJ Associate Professor of Chulalongkorn University, Bangkok, Thailand

Acknowledgements

I would like to thank the people who helped and supported me during my Ph.D. studies.

Firstly, I would like to express my sincere gratitude to my advisor Professor Takasu Atsuhiro for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge.

I want to thank all the members and internship students in Takasu Laboratory, especially Takenaka Akiko, for providing a motivating, fun, and positive working environment. Also, thanks to the people in NII for all the support and fruitful discussions about both research and life in Japan.

Last but not the least, I would like to thank my family: my parents and to my brother and sister for supporting me spiritually throughout writing this thesis and my life in general. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Abstract

Recently, as the amount of information created by people has increased, users found it difficult to choose items that would best suit them from among the several options available. Therefore, filtering tools that can recommend appropriate items to users become important. Recommender Systems (RSs), have been invented and become a key approach to solve the information overload problem. RSs aim to provide personalized recommendations on specific items to the individual users based on the users' individual tastes.

Collaborative Filtering (CF), one of the most popular RSs techniques, usually considers the influence of neighbors (i.e., users and items) when recommending suitable items for a target user. CF can be divided into two approaches: neighborhood-based approach and model-based approach. Both of them have their own advantages and disadvantages. In this thesis, a hybrid CF model is proposed to combine advantages from neighborhood-based and model-based together while overcoming their disadvantages. In order to propose a hybrid CF model, the first challenge is how to efficiently combine neighborhood-based and model-based models together. Moreover, the problems in both neighborhood-based and model-based approaches need to be solved.

Since the purpose of this thesis is to apply the neighborhood-based approach concept into the model-based approach, the first challenge can be considered as how to utilize neighbors into a hybrid CF model. Information from neighbors can be in many forms, but in this thesis, I focus on explicit feedback, which can capture the different level of interests from users better than implicit feedback. By using explicit feedback in the hybrid CF model, a novel user and item representation is proposed. Unlike traditional embeddings, this thesis proposes user representation matrix and item representation matrix which represent user and item in a dense matrix instead of a single vector like in most of existing works. These novel user and item representations are used to store users and items characteristics which a single vector cannot and then available for the next purpose which is a rating conversion.

Second, in the neighborhood-based approach, based on real-world assumptions, different neighbors have different influence levels on the target user. Thus, I propose a module to select high–quality neighbors (i.e., friends and historical items), instead of randomly selecting

them. Additionally, most neighborhood-based models use the actual ratings from neighbors to predict the ratings of the target user toward target items without considering the difference between their rating patterns, which often leads to a low accuracy prediction caused by the *improper rating-range* problem. Recently, rating conversion methods have been proposed to address this issue. Because each user can have a different level of influence on a pair of target user–item, I propose a *friends* module, which converts their ratings to match the target user's perspective and assigns different weights to each user before modeling latent relations and predictions. Moreover, since item information is as important as user information, I also propose an *items* module, which utilizes the target user's historical items information to help predict how much target user would like the target item.

Third, concerning a model-based approach, one point to consider is how to model the relationship between a pair of user–item and how to optimize the model. In order to keep information of user–item pair as much as possible, I also learn latent relation vectors between each user and item by adopting *knowledge graph* embedding ideas, which can model latent relation between a specific pair of user–item, also can alleviate the geometric inflexibility that occur in [39] which they try to put a pair of user and item at the same point in the vector space.

Finally, all components are combined to build the hybrid CF neural network (NN) model named *attentive hybrid collaborative filtering for rating conversion in recommender systems* (AHCF). The FilmTrust and MovieLens datasets are used in experiments comparing the proposed method with state-of-the-art methods. The experimental evaluations have shown that my proposed model is more effective than existing methods.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

Recommender systems (RSs) are the systems that were invented and developed to provide a personalized recommendation on the suitable items that would be satisfied by an individual user. In recent years, as the amount of information created by people's daily activities has increased, it causes an explosion of the information in the modern age. Users have found it challenging to select products that best suit them from among several options available. As a result, RSs have been developing over the years to solve the problem of information overload. Nowadays RSs have become an important engine for many platforms, e.g. e-commerce, online news, and social network sites (SNS). The key to RSs is in suggesting items to individual users in hope that recommended items will be satisfactory for the users.

There are many approaches in RSs. The most prevalent and popular approach for recommending items to a *target user* (the user to whom recommendations are targeted) is the collaborative filtering (CF) approach, which is based on the similarity of users or items to previous interactions. It utilizes historical interaction (e.g., clicks, rates, purchases, etc.) to infer the user's preference and recommend items based on the matching score between the target user and the target item.

Typically, CF can be divided into two approaches: a neighborhood-based approach and a model-based approach. Although the neighborhood-based approach is simple and works reasonably well in practice, it requires a high cost in terms of computing time and spatial complexity. Also, it can suffer from sparsity problems, which is one of the challenges in the RSs field. Therefore, nowadays, a model-based approach has gained more popularity than a neighborhood-based approach because it is more capable of handling the problem of sparsity and scalability. However, the model-based approach requires a great resource to develop the

model and may lose information when performing dimensionality reduction. In addition, the model-based approach still suffers from interpretability and lack of explainability.

From the above-mentioned paragraph, it indicates that both the neighborhood-based approach and model-based approach provide both advantages and disadvantages. In this thesis, I would like to build a hybrid CF model that leverages and preserves the advantages of both neighborhood-based and model-based approaches while overcoming their disadvantages. The challenge of this thesis is how to build and propose a powerful hybrid CF model which can maintain the advantages of both CF approaches and reduce their disadvantages. Therefore, this thesis focuses on solving the problems in both the neighborhood-based approach and model-based approach, and focuses on how to combine them into the hybrid CF model.

In the neighborhood-based approach, people in the real world tend to believe their friends' opinions when making decisions. Therefore, in CF RSs, the target user's friends might have an impact on the target user when making decisions on items. Based on this real-world assumption, RSs utilize and leverage the opinions of friends to predict how much the target user would like an item.

Typically, the neighborhood-based approach utilizes user rating data to compute the similarity between users or items. In order to predict the rating score of the user toward the target item, CF RS aims to identify the set of the target user's friends and use their actual rating scores to determine how much the target user would like the target item. There are several approaches to finding a set of friends. In case of there is no explicit friendship relation in the dataset, most RSs often aim to find the set of users who have rated the target item in the past or have mutual rated items with the target user. Then, the predicted rating score for the target user toward the target item is then calculated using the actual scores of friends. However, utilizing the actual rating scores from friends directly often leads to low-accuracy predictions because of the *improper rating-range problem* [15].

The improper rating-range problem occurs when the range of rating patterns of each user is different. Because each user has a unique rating pattern, a rating score needs to be interpreted. Even if two users give the same item the same score, it does not always indicate they like it to the same extent if their rating patterns are different. As a result, utilizing the actual ratings from users who rate items within different ranges to predict the rating score of the target user is ineffective and may result in low recommendation accuracy. Therefore, some researchers have adjusted the ratings from several ranges to match the common range in advance before using them in the prediction step [15, 44, 43, 50]. These methods can be called *rating conversion*.

Aside from this improper rating-range problem, there is one more problem to consider because each of the target user's friends is likely to have a different level of influence on the

target user. In the real world, everyone has biases. It is common to say that all friends are not equal, with user *A* having more or less influence on the target user than user *B*.

Therefore, there should be a module that helps the model in modifying each friend's rating score to match the target user's perspective before using their scores in the calculation. This can address the issue of improper rating range. Then, to account for each friend's differing influence level, this module will assign an individual weight to each user based on the relation between each friend and the target user–item pair.

For neighborhood-based CF and for such rating conversions, the most essential input is ratings that involve explicit feedback, which is the input of users regarding their interest in an item. This is the most helpful information since it directly comes from the user and shows their direct interest regarding the item. For evaluating the performance of RSs, there are two popular approaches, which are *rating prediction* and *item ranking*. *Item ranking* has recently become more popular because some datasets contain implicit feedback and most of recent works focus on implicit feedback [37, 70]. Implicit feedback is the interaction between a user and an item that does not have a rating score (i.e., click, buy, rate, etc.). However, some methods have treated explicit feedback as implicit feedback and input them into their model [24, 37, 55, 70] which I believe would be preferable if the model could leverage and make the most use of the data in the form of explicit feedback rather than transforming it into the implicit feedback form. Because explicit feedback or ratings are more expressive and more powerful than implicit feedback. Each rating score might imply a different meaning. In reality, just because a user rates an item does not indicate the user likes that item. For example, if user *A* gave a 1-score to item *I*, it does not mean that user *A* like item *I*. User *A* might dislike item *I*, so he/she gave only 1-score. Therefore concrete ratings are important for prediction in RSs.

Although some works use explicit ratings in their neural network (NN) models, they only use ratings as a label for optimization [3, 14, 21]. Those ratings were not directly incorporated as input and ratings did not play an important role in their model architecture. Also, their user and item representations are usually in the form of one-hot vectors which do not indicate specific characteristics of user's ratings or item's ratings. This thesis aims to incorporate ratings as an input of this end-to-end CF model and specify the interaction between a pair of user–item in more detail by rating information.

On the other hand, for the model-based approach, even though some works have shown that it can provide more accurate results than the neighborhood-based approach, it still has some flaws and weaknesses.

According to the model-based approach, the modeling process is usually conducted by machine learning techniques. Recently, some model-based CF models have usually been

implemented based on NN and deep learning. Collaborative metric learning (CML) [39] is one of the model-based approach CF that try to assign a user–item pair to the same location in a vector space by reducing the distance between each user–item interaction, in which their scoring function is geometrically restrictive [70]. Because their model tries to fit a user and all his interacted items onto the same point, it causes geometrically congestive and inflexible. This thesis tries to solve such a problem by utilizing the knowledge graph (KG) embedding approach to learn the latent relation vector between user–item pairs instead of trying to put them into the same point in the vector space.

1.2 Problem Definition

According to the motivation in the previous subsection, this thesis focuses on building a hybrid CF which includes neighborhood-based approach and model-based approach. Therefore, there are three main problems to consider which are: how to combine neighborhood-based CF and model-based CF together, problems in the part of neighborhood-based approach, and problems in the part of model-based approach. All of the sub-problems in three approaches are listed as follows.

1.2.1 Hybrid Collaborative Filtering: How to apply neighborhood-based approach to model-based approach

The first challenge of this thesis is how to apply neighborhood-based approach to model-based approach. This challenge can be considered as how to utilize neighbors into hybrid CF models.

Utilizing Neighbors in Hybrid CF: Incorporate explicit feedback in an end-to-end hybrid CF model

Nowadays, most model-based CF RSs utilize implicit feedback to be an input to their model. If the dataset contains implicit feedback, that is understandable and acceptable. However, some of them utilize datasets that contain explicit feedback, but treated those explicit feedback as implicit feedback (i.e, binary score) and utilize implicit feedback in their models [24, 37, 55, 70] because those models cannot handle explicit feedback. The problem is that converting explicit feedback into implicit feedback might make a rating score loses its impactful meaning. For example, if user *A* rated different items with different rating scores, it means user *A* like those items at the different levels. In reality, just because a user interacts with an item does not indicate the user like that item. Therefore, explicit feedback should

stay in its original form because specific rating scores are more powerful and can express more meaning than implicit feedback.

Despite the fact that some works utilize explicit feedback in their model-based CF model, they only use ratings as a label for optimization of the model [3, 14, 21]. Those explicit feedback were not directly incorporated as input and were not embedded in user and item embeddings, so those explicit feedback did not play an important role in their model architecture. Furthermore, their user and item representations are usually represented in the form of one-hot vectors which cannot indicate or prove specific characteristics of user's and item's ratings.

According to the above reasons, concrete rating scores are important for prediction in RSs. The goal of this thesis is to incorporate explicit feedback into an end-to-end model in which those explicit feedback can specify the characteristics of users and items and can specify the interaction between user–item pairs in more detail. Therefore, in this thesis, I propose how to incorporate explicit feedback in the hybrid CF model.

1.2.2 Solving Problems in Neighborhood-based CF Approach

All neighbors are not equal

In the neighborhood-based CF model, systems usually use the opinion of the target user's friends (i.e., neighbors who have rated the target item in the past) and use their opinions to generate recommendations for the target user. In the real world, each friend has a different influence on the target user. Naturally, people tend to believe their close friends when deciding to choose something more than their regular friends. Therefore, this indicates that all friends are not equal.

Because I would like to directly utilize the concept of neighbors in the model-based CF model, this problem should be carefully considered. This hybrid CF should consist of a module that can select the high-quality neighbors to have more contribution to the model than the low-quality users and items.

Rating Conversion between a pair of users

According to the improper rating-range problem in the neighborhood-based CF approach mentioned in the previous subsection, the actual ratings of neighbors should not be used directly in the prediction step. They should be converted into the target user's perspective first using rating conversion methods.

Most rating conversion methods [4, 15, 50] always convert ratings between a pair of users in the neighborhood-based CF which is simple and easy to implement. However, a

rating conversion in a hybrid CF model, which combines both neighborhood-based CF and model-based CF, is one of the main challenges of this thesis.

Because the main idea is to apply the neighborhood-based concept to model-based CF, so the rating conversion should be considered on the basis of model-based CF, in which user, item, and rating representations are usually in embeddings form. Unlike other rating conversion methods which try to convert rating in scalar form, this thesis proposes how to do rating conversion in vector form within the model-based CF.

1.2.3 Solving Problems in Model-based CF Approach

Applying Knowledge Graph Embedding concept to a hybrid CF model

One thing to consider in model-based CF is how to model user–item relationships. Recently, some model-based CF methods try to construct user–item relationships in several ways. Collaborative Metric Learning (CML) [39] proposed a metric-based learning scheme that minimizes the distance between user and item vectors of positive interactions. This can learn user–user similarity and item–item similarity in vector space. However, this method faces several weaknesses. First, according to its scoring function, it tries to fit a pair of user–item into the same point in vector space. This might be effective for *one–to–many* relationships and small datasets, but for *many–to–many* relationships and large datasets, this method is quite not suitable because the optimal point of each user and item will be placed at the same point in the space. Intuitively, this tries to fit a user and all his interacted items onto the same point, i.e., geometrically congestive and inflexible.

In order to solve this problem, LRML [70] introduces a new way to construct a latent relation vector between a pair of user–item by adopting the idea of *TransE* [12], a translation-based embedding model that models the relationships between entities in a KG. Instead of trying to fit a pair of user–item into the same point in the vector space as CML does, LRML learns an optimal translation between each user–item interaction.

Another reason for introducing the KG embedding technique is that I believe the latent relation vector can help the model to learn better user and item embedding, which can lead to better recommendations. Because a vector value often has more expressive power than a scalar value, so we can expect to obtain higher quality embedding. Additionally, this latent relation vector is highly personalized because it is generated based on each user–item pair, so each user–item pair will have its own latent relation vector.

1.3 Objective

According to the problem definition in the previous subsection, this thesis is proposed under the following three main objectives.

- **Utilizing neighbors and incorporating explicit feedback in the hybrid CF model**: with the purpose of applying the neighborhood-based CF concept into the model-based CF model, neighbors (i.e., friends and historical items) should be included in this proposed hybrid CF model. Additionally, those neighbors contain information in the form of explicit feedback, so explicit feedback should be used in this model. In order to make the most use of explicit feedback, explicit feedback should be used properly in the end-to-end CF model with respect to the following properties. First, explicit feedback should not be transformed into implicit feedback like in some well-known methods. Second, explicit feedback should be embedded properly in user and item representations, which can specify characteristics of users and items, instead of using it only for optimization.

- **Neighbors selection and rating conversion between a specific pair of users in the end-to-end hybrid CF model**: because the neighborhood-based approach can be divided into two approaches: user-based CF and item-based CF, thus neighbors should come from both user side and item side. Moreover, neighbors should be carefully selected instead of randomly selected because all neighbors provide different influences on the target user and the target item. The model must select high-quality neighbors to participate in more role in the model. For rating conversion, unlike other methods that can convert only a rating score in a scalar form, this proposed rating conversion method must allow performing on vector value instead of scalar value. In addition, this rating conversion must perform in the end-to-end hybrid CF model.

- **Applying knowledge graph embedding concept to the hybrid CF model**: in order to get rid of geometric inflexibility in CML, the optimal translation of interaction between a specific pair of user–item should be learned properly. This translation of interaction is a latent relation vector. It must be specific to each user–item pair.

1.4 Contribution

This thesis is composed of three main topics which contribute to the previously mentioned objectives. These contributions are proposed and presented in Attentive Hybrid Collaborative

Filtering for Rating Conversion in RSs (**AHCF**) [74]. Note that I have proposed Translation-based Embedding Model for Rating Conversion in RSs (**TransRS**) [73], and this TransRS consists of three main topics listed below but in a different way with AHCF [74]. Moreover, for the topic of neighbors selection and rating conversion, I have proposed methods in Trust-based Ant RSs (**TrustAnt**) [72]. The details of them are described in each chapter of this thesis.

1.4.1 Utilizing Neighbors in the End-to-end Hybrid CF

First, to utilize neighbors (i.e., friends and historical items) in the end-to-end hybrid CF model, novel user and item representations are proposed. Unlike traditional model-based RSs which represent user and item in embeddings, this proposed novel user and item representations (i.e., *UR*-matrix and *IR*-matrix) represent user and item in a form of a dense matrix in which each row of matrix contributes to rating scores.

The experimental results have shown that my proposed UR-matrix and IR-matrix can help the model provide better prediction results than the traditional user and item representations.

1.4.2 Neighbors Selection and Rating Conversion in the End-to-end Hybrid CF

Second, instead of randomly selecting neighbors, the neighbors selection module is proposed (i.e., friend selection and historical item selection) to select high-quality friends and historical items to participate more role in the model than the low-quality one. Moreover, the rating conversion method between a pair of users is proposed to solve the improper rating-range problem. Unlike the existing rating conversion methods that convert ratings in scalar form, this rating conversion method in the friend module can convert ratings between users in embedding vector form within the end-to-end NN model.

The extensive experimental results have shown that with the neighbor selection module, friend module, and item module, the model can provide better prediction results than the existing work.

1.4.3 Incorporating Knowledge Graph Embedding to Recommender Systems

Third, I introduce the KG embedding idea to the end-to-end CF model in order to alleviate the geometric inflexibility in CML [39] method which tries to place the observed pair of user

and item at the same point. Inspired by LRML [70], TransE [12] technique is adopted to be a scoring function in this hybrid CF end-to-end NN model.

1.4.4 Integration of Proposed Models

Finally, the integration of TrustAnt [72], TransRS [73] and AHCF [74] are described in detail. This chapter shows how to construct each model and shows the experimental evaluation of each model. The discussion shows that all works can address the previously mentioned problem properly. However, AHCF [74] is more efficient because of its end-to-end architecture, which is more practical than the TransRS [73] which is building blocks architecture, and more practical than the TrustAnt which is the neighborhood-based CF RSs.

1.5 Thesis Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 introduces the background of recommender systems, ant colony optimization, knowledge graph embedding, rating conversion, and attention mechanism in deep learning. Chapter 3 presents the related techniques to this work, including the existing RSs, knowledge graph embedding-based RSs, and rating conversion techniques. From Chapter 4 to Chapter 7, the details of both TransRS [73] and AHCF [74] will be described in detail based on each topic. Chapter 4 explains how to utilize neighbors into an end-to-end hybrid CF model. Chapter 5 presents how to select neighbors and how to perform rating conversion between a pair of users. Chapter 6 presents how to incorporate KG embedding technique to RSs. Chapter 7 demonstrates the contribution of the models and experimental evaluations of each model. Chapter 8 finally summarizes the thesis and discusses future work.

Chapter 2

Background

2.1 Recommender Systems

Due to the explosion of the information online in the modern age, recommender systems (RSs) have been developing over the years to solve the problem of information overload. RSs have become an important tool for many platforms e.g. e–commerce, online news, and SNS.

The goal of RSs is to suggest and recommend products or items to an individual user in hope that the recommended item will satisfy the user [1, 62, 63]. RSs have been applied in many domains, including movie, music, news, etc. The traditional recommendation approach is categorized into three main approaches, namely the collaborative filtering (CF) approach, the content-based approach, and the hybrid approach.

Most RSs rely on interactions between users–user, users–items, and items–items. They use these interactions as input for producing personalized recommendations. Such interactions can be in various formats, such as clicks, likes, and purchases, but the most common form is the numeric ratings. For example, Figure 2.1 shows the user rating system from Amazon.com, where users can provide ratings to their purchased products with a score in the range of 1 to 5. These ratings indicate the user preference levels toward those items. Commonly, the higher the score they give towards items, the higher their satisfaction towards those items. Note that rating range 1–5 of Amazon.com is one example of the rating system, the range of rating scores can be varied on the different systems. For example, Agoda.com and Booking.com, the search engines for travel and accommodation, use a 10-scale rating system where users can provide a rating from 1 to 10 to their visited properties on various criteria. These ratings are used as the main input for the recommendation algorithms for making personalized recommendations for individual users based on the predicted rating score of unseen items.

Figure 2.1: A user rating system from Amazon.com.

2.1.1 Collaborative Filtering

The most popular and widely utilized technique is the CF approach. It recommends items to individual users based on their neighbors (i.e., users and items). Typically, CF is further divided into the *neighborhood-based* approach and the *model-based* approach.

Neighborhood-based approach

The neighborhood-based approach, or it can be called the memory-based approach, is a traditional CF technique that utilizes user or item information in the recommendation step in which user–item ratings are stored in the system and used directly to predict ratings for new items. The neighborhood-based approach has several advantages, including simplicity, justification, and stability [25].

The neighborhood-based approach can be further divided into two types: *user-based* CF which the predicted rating of the target user towards the target item is calculated depending on the target item's ratings by other similar users and *item-based* CF which the target item's rating is predicted based on how similar items have been rated by the target user. In this thesis, I utilize both types of neighborhood-based CF approaches.

The neighborhood-based approach can be further divided into two types: user-based neighborhood-based CF and item-based neighborhood-based CF. To simplify, I will use an user–based approach and an item–based approach throughout the entire thesis. The details of each type are defined as follows:

- **User-based approach**: in the user–based CF approach, given a target user and target item pair, the predicted score of the target user towards the target item is calculated depending on the target item's ratings by other similar users. In this approach, the predicted ratings are predicted using the ratings of neighboring users. The neighborhoods are defined by similarities among users.

- **Item-based approach**: opposite to the user–based CF approach, in the item–based CF approach, given a target user and target item pair, the predicted score of the target user toward the target item is calculated based on how similar items have been rated by the target user. In this approach, the predicted ratings are predicted using the target user's own rating behavior on neighboring (closely related) items. The neighborhoods in this approach are defined by similarities among items

Model-based approach

Although the neighborhood-based approach is simple to implement and works reasonably well in practice, it comes at a high cost in terms of computing time and spatial complexity, and it can also suffer from sparsity problems. Therefore, the model-based approach, which learns a predictive model using the user–item ratings, is now gaining popularity. Its main advantages over the model-based approach are scalability, prediction speed, and avoidance of overfitting. Therefore, most recent CF models adopt the model-based approach to their models.

Model-based approach is a recent trend in RSs where machine learning and data mining methods are used in the context of a predictive model. Recently, the model–based approach is more popular because of its scalability and faster than the memory–based approach. The quality of predictions may not be as accurate as with the memory–based approach because the model–based approach is not using all the information available (depending on the way the model is built).

Hybrid Collaborative Filtering

The term 'hybrid' in general refers to 'combining different things' or 'composed of different elements'. In RSs, the term hybrid RSs refers to RSs that combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one [13]. However, the term hybrid CF can refer to several kinds of combinations.

First, hybrid CF can refer to CF that combines one of CF techniques (i.e., neighborhood-based CF or model-based CF) with side information e.g., review, trust, etc. Second, hybrid CF can refer to CF that combines user-based and item-based neighborhood-based CF approaches.

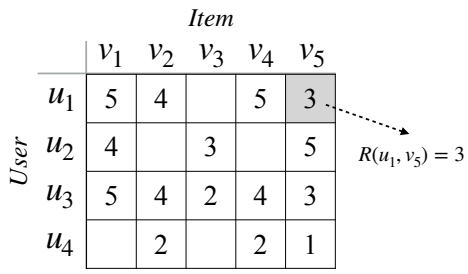

Figure 2.2: A user-item rating matrix.

Another one is hybrid CF which combines neighborhood-based CF and model-based CF approaches together. In this thesis, I focus on the last one, a hybrid CF which refers to the CF that combines neighborhood-based CF and model-based CF approaches together.

In this thesis, I propose a hybrid CF network for RSs that combines the advantages of both the neighborhood-based CF (containing both user-based CF and item-based CF) and model-based CF approaches.

Recently, research in RSs field is rapidly being developed by researchers all over the world. The key challenge of RSs is to recommend the right item to individual user based on their interest. Therefore, user's interest or user's preference is the most important thing to consider to build good recommendations. Over the past few years, several researchers have tried to find new ways to represent user's preferences. Some have proposed that Translation-based embedding model is one of the ways to represent the user's preferences in the graph form.

2.1.2 Basic Recommendation Strategies

After the ratings are collected, they can be represented by a user-item rating matrix, as shown by the example in Figure 2.2. Each row and each column in this matrix respectively represent each user and each item in the system, whereas each entry of the matrix represents the corresponding rating of a user given to an item. In Figure 2.2, for example, the user u_1 assigned a rating score "5" to the item i_1. The main task of a standard recommender system is to predict the ratings of items that are not yet rated by the individual users, i.e. those denoted by empty entries in Figure 2.2.

Formally, given $User$ a set of users, $Item$ a set of items, and $Rating$ a set of possible rating values (e.g. 1 to 5), an objective function R of a standard recommender system can be defined by:

$$R : User \times Item \rightarrow Rating. \tag{2.1.1}$$

After the ratings are computed, the items retrieving the highest prediction scores are selected as the ones to be recommended to an individual user, implying they are highly relevant to his/her personal preferences.

Based on the CF approach described in the previous subsection, the model implemented by the CF approach relies on the ratings from users who shared similar interests and preferences for generating a rating prediction. From Figure 2.2, for example, the user u_3 is very similar to user u_1 since they provided similar ratings to similar set of items. Therefore, the ratings from user u_3 on items that do not yet rated by u_1 (e.g. v_3) can be helpful for predicting the rating for u_1. In this case, u_3 can be considered as a neighbor or friend of u_1. The simplest method in a neighborhood-based CF approach is the k-nearest neighbors (k-NN) technique [1], which makes a prediction by incorporating the ratings from the top k most similar neighbors to the target user. This method first computes the similarity between every pair of users by using a similarity metric such as Pearson correlation coefficient, as expressed by

$$sim(u_a, u_b) = \frac{\sum_{v_j \in I(u_a,u_b)} (r_{u_a,v_j} - \bar{r}_{u_a})(r_{u_b,v_j} - \bar{r}_{u_b})}{\sqrt{\sum_{v_j \in I(u_a,u_b)} (r_{u_a,v_j} - \bar{r}_{u_a})^2} \sqrt{\sum_{v_j \in I(u_a,u_b)} (r_{u_b,v_j} - \bar{r}_{u_b})^2}}, \tag{2.1.2}$$

where $sim(u_a, u_b)$ denotes the similarity between users u_a and u_b, $I(u_a, u_b)$ is the set of items rated by both u_a and u_b, and $\bar{r}_{u_a}$ is the average rating of u_a. The rating of the user u_a on an unseen item v_k can then be estimated by

$$\hat{r}_{u_a,v_k} = \frac{\sum_{u_b \in Neighbors(u_a)} (r_{u_b,v_k} - \bar{r}_{u_b}) \times sim(u_a, u_b)}{\sum_{u_b \in Neighbors(u_a)} |sim(u_a, u_b)|}, \tag{2.1.3}$$

where $Neighbors(u_a)$ denotes the set of k most similar neighbors of user u_a. To produce an accurate prediction, the neighborhood-based approach requires a significant amount of mutually-rated items by users, in order to identify high-quality neighbors.

On the other hand, the model-based approach exploits the rating data to build a predictive model, and uses it for future rating prediction. Many model-based CF techniques have been proposed, among them the latent factor model [49] is the most popular one due to its effectiveness in delivering highly accurate predictions. The standard method in latent factor

model is the matrix factorization (MF) technique [49] that models a rating as an interaction between the latent features of user and item. Specifically, each user u_i and each item v_j are respectively associated with user and item latent-feature vectors $\mathbf{x}_{u_i}$ and $\mathbf{x}_{v_j} \in \mathbb{R}^k$, where k is the number of latent dimensions. The rating of user u_i for item v_j is then estimated by

$$\hat{r}_{i,j} = \mathbf{x}_{u_i}^T \mathbf{x}_{v_j}. \tag{2.1.4}$$

The parameters $\mathbf{x}_{u_i}$ and $\mathbf{x}_{v_j}$ are learned and optimized by minimizing the regularized square error through a loss function defined by

$$L = \sum_{(i,j)\in O} (r_{i,j} - \hat{r}_{i,j})^2 + \lambda(\|\mathbf{x}_{u_i}\|^2 + \|\mathbf{x}_{v_j}\|^2), \tag{2.1.5}$$

where O denotes the set of observed user–item rating pairs, $r_{i,j}$ is the observed rating score of user u_i toward item v_j, and λ is a constant controlling the regularization rate. (The regularization term is added to avoid overfitting the observed rating data.)

The parameters $\mathbf{x}_{u_i}$ and $\mathbf{x}_{v_j}$ can be considered as the representations of user u_i and item v_j, respectively, in the latent space. Because they are learned and optimized from observed ratings, their representation quality depends on the quantity of available historical ratings. For most rating datasets, however, users typically rate only a few items among all the available items in the system, which leads to a rating sparsity problem. Such a problem would directly affect the quality of the user and item representations.

2.1.3 Challenges in Recommender Systems

In the past, content-based filtering was one of the most used approaches in RSs [8, 66]. However, nowadays CF is the most widely used approach in RSs. In this thesis, I focus on CF RSs, so this subsection will show some of related trends and challenges in RSs.

In order to provide the effective recommendations, there are two common challenges that RSs aim to achieve: focusing on accuracy and the effectiveness of RSs, and dealing with sparsity data.

Focusing on Accuracy and the Effectiveness of Recommender Systems

Most research papers in the RSs field focus on accuracy and the effectiveness of the RSs [8]. Because the goal of RSs is to bring the most satisfying items to individual user, researchers

seem to assume that an accurate RSs will lead to high user satisfaction. Therefore, the vast majority of RS methods measure the effectiveness of their recommendation algorithms based on the prediction accuracy.

The standard RS strategies such as the CF–based approach, which utilize only the rating data between users and items for making a recommendation, often fail to deliver highly accurate predictions. The accuracy of a neighborhood-based CF approach relies heavily on the quality of the neighbors who have rated the target items. Therefore, *neighbors selection* is one of the challenges in neighborhood-based CF. Several efforts tried to enhance the computation of similarity [38, 54]. For example, [38] introduced the weighted Pearson correlation coefficient by modeling the confidence level among the neighbors. [54] proposed a novel similarity measure by considering the proportion of common ratings between two users.

By building a predictive model for future rating prediction, the model-based CF approach, in contrast, is more scalable than the neighborhood-based CF approach. However, the prediction made by the model-based CF approach might not be as accurate as the neighborhood-based CF approach when the user–item rating matrix is dense (i.e., when there is a sufficient amount of ratings to identify high-quality neighbors). Therefore, several model-based CF methods have attempted to invent models with more accurate predictions. For example, many variants of MF have been proposed to improve the prediction accuracy upon the baselined MF [40, 49, 51, 65]. Recently, a neural network model has been integrated into a latent factor model to further enhance the prediction accuracy [37].

By considering the advantages of both neighborhood-based and model-based CF approaches, this thesis aims to build a hybrid CF model which leverages and preserves the advantages of both neighborhood-based and model-based CF approaches. Therefore, problems and challenges in both neighborhood-based and model-based CF approaches are need to be solved. Moreover, one more challenge is how to build a powerful hybrid CF model which can maintain the advantages of both CF approaches and reduce their disadvantages.

Dealing with Data Sparsity

In many large-scale RSs, there are a large number of users and items. However, in most of them, many users only provide ratings to a few items, compared to all available items, and many items might be rated only a few times by the users. Representing this kind of data with the user-item rating matrix would consequently result in a sparse matrix where most of its entries are missing. This leads to one of the major challenges in RS, called *data sparsity* problem, which directly impacts the predictive performances of many recommendation algorithms.

Since a CF approach relies on historical ratings for future rating prediction, it often suffers from a rating sparsity problem, where there are insufficient amounts of ratings for producing an effective recommendation. For example, the neighborhood-based CF approach relies on the overlapped ratings on the mutually rated items between two users to compute their similarity. With the sparse user-item rating matrix, such overlapped ratings might be very few or unavailable, so it makes the computation of user similarity unreliable [22, 33]. This leads to the ineffective process of identifying the high–quality neighbors, and consequently reduces the prediction accuracy.

Also, the predictive performance of the model-based CF approach is also affected by the sparsity problem. By building the predictive models on the sparse rating data, those models might be trained to over-fitting with the only available user-item pairs. In spite of providing high accuracy on the training (already seen) data, the over-fitted model tends to provide poor accuracy on the unseen data.

Many works in RSs field have attempted to alleviate the sparsity problem. For example, [32, 41] combined content-based recommendation approach with the CF-based approach by considering similarity of the item characteristics. Moreover, some works applied the dimensionality reduction techniques such as the singular value decomposition (SVD) [48], the principle component analysis (PCA) [67], etc., to overcome the rating data sparsity. For example, [31] used predicted ratings of SVD to fill in missing values in the user-item rating matrix and then applied the traditional item-based CF to make a recommendation.

In this thesis, I would likm3e to combine the neighborhood-based CF and model-based CF together to overcome the data sparsity problem by utilizing neighbors, both users and items, into the hybrid CF model.

2.2 Ant Colony Optimization in Recommender Systems

Ant colony optimization (ACO) is a swarm intelligence method which was proposed by Dorigo in 1992 [28]. ACO is a probabilistic technique for solving optimization problems that can be reduced to find a good path through graphs. The ACO process is based on the foraging behavior of real ants in a colony when they are seeking a path for a good food source. With this concept of ACO, some researchers in the RSs field have applied ACO to RSs.

In RSs, ACO is usually applied to trust-based RSs, which is one kind of CF RSs that rely on the concept of trust between users, rather than similarity. In trust-based RSs, the model usually finds neighbors based on trust value.

In RSs terms, the artificial ants are sent out from the target user into the network to search for neighbors (i.e., reliable users who would be expected to provide good recommendations)

just as real ants search for good food sources. The key mechanism from the ant colony (AC) concept is a 'pheromone-update' mechanism. The artificial ants traverse paths and update the pheromone level on the edges as real ants do.

There are several approaches of ACO and two main ACO techniques have been applied in trust-based RSs. First, the traditional ant colony (AC) concept, in which the pheromone value will be updated after all ants have completed each iteration. One iteration means that all ants in the network have completed their constructed paths, which will include a number of nodes and edges. Bedi, P. et al. [6] and Kaleroun, A. et al. [46] implemented the traditional AC concept in their works.

Second, the ant colony system (ACS) represents an improvement on the original ACO. The ACS concept differs from the original in that the pheromone-update strategy is divided into two parts. The first path involves a local pheromone update, in which a pheromone value is updated locally after each step of an ant. This means that the pheromone value on an edge that the ant chooses to cross will be updated. The second part involves a global pheromone update, in which the pheromone value will be updated after all ants have completed an iteration. When all ants have completed one iteration, the pheromone value on some or all paths will be updated. The paths selected for updating depend on the optimization conditions adopted by the particular system being proposed. Bellaachia, A. et al. [9] implemented ACS in their work.

In this thesis, I adopted the ACO concept and applied to trust-based RSs in order to find high-quality neighbors by mimicking the real ant behavior.

2.3 Knowledge Graph Embedding

Knowledge Graph (KG), also known as a semantic network, is a multi-relational graph that represents a real-world entities (i.e., objects, events, situations, or concepts) and illustrates the relationship between them. A KG is made up of three main components: nodes, edges, and labels. Any object, place, or person can be a node (i.e., entities). An edge defines the relationship between the nodes. For example, a node could be a movie, like *Titanic*, and a director like, *James Cameron*. An edge would be to categorize the relationship as a *direct* relationship, which James Cameron directed in the Titanic movie.

In recent years, KG construction and application gain popularity and a large number of KGs, such as Freebase, DBpedia, etc. have been created and successfully applied to many real-world applications [76]. KGs usually store entities and relations in a triple of the form (head entity, relation, tail entity), also called a *fact*, indicating that two entities are connected by a specific relation. The previous example, which explains the relationship between

Titanic movie and Jame Cameron, can be represented in a form of (Tinatic, directed_by, JamesCameron). In spite of the effectiveness in representing structure, the underlying symbolic nature of such triples makes KGs hard to manipulate.

To address this issue, a knowledge graph embedding (KGE) has been proposed and quickly gained vast attention. The key idea of KGE is to embed components of a KG including entities and relations into continuous vector spaces, in order to simplify the manipulation while maintaining the inherent structure of the KG.

KGE can be roughly divided into 2 groups: Translation-based embedding model and Semantic matching model. The details of each group are described as follows.

2.3.1 Translation-based Embedding Models

Translation-based embedding models are widely used in the KG field. KG embedding is to embed components of KG including entities and relations into continuous vector space. This translation in vector space takes inspiration from Natural Language Processing (NLP) field.

Recently, many KG embedding techniques have been proposed and rapidly gain numerous attention [76]. TransE [12], which is proposed by Bordes et al., is the simplest and the most representative translation-based embedding model. It represents both entities and relations as vectors in the same space. The scoring function of TransE is defined as the distance between $h+r$ and t, where h refers to head entity, t refers to tail entity, and r refers to relation between head and tail entities.

TransE is one of such models and the first translation-based method. TransE is well known because the principle of TransE can effectively capture the rules of a knowledge graph although it seems very simple. In spite of its efficiency and simplicity, TransE has flaw in dealing with 1-to-N, N-to-1, and N-to-N relations. Therefore, various translation-based embedding models have been proposed and developed over the past few years, e.g. TransH [78], TransR [53], TransD [42], etc. [76] Figure 2.3 shows the illustration of different translation-based embedding models.

2.3.2 Semantic Matching Model

Unlike translation-based embedding models, semantic matching models exploit similarity-based scoring functions. They measure plausibility of facts by matching latent semantics of entities and relations embodied in their space representations. There are several KGE semantic matching models, but in this thesis, I focus on the concept of translation-based embedding model (i.e., TransE), so this subsection will briefly describe their properties.

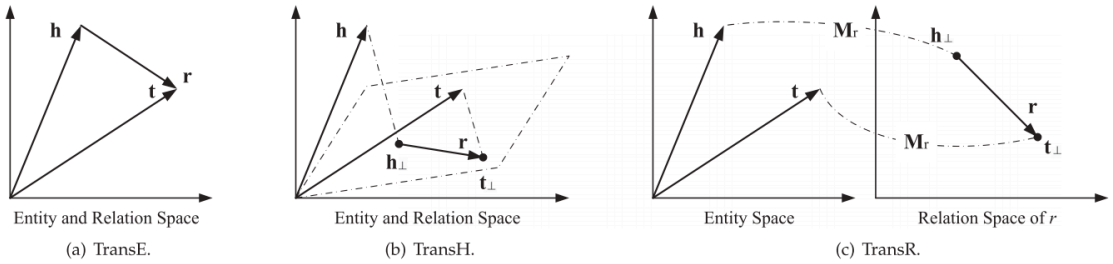

Figure 2.3: Simple illustration of Translation-based Embedding Models (i.e., TransE, TransH, and TransR). Adapted from [76]

RESCAL [58] connects each entity with a vector to capture its latent semantics. Each relation is represented as a matrix that models pairwise interactions between latent factors. The score of a fact (h,r,t) is defined by a bilinear function.

$$f_r(h,t) = \mathbf{h}^T \mathbf{M}_r \mathbf{t} \tag{2.3.1}$$

This score captures pairwise interactions between all components of h and t.

DisMult [79] simplifies RESCAL by restricting M_r, to diagonal matrices. For each relation r, it introduces a vector embedding $\mathbf{r} \in \mathbb{R}^d$ and requires $\mathbf{M}^r = diag(\mathbf{r})$. The scoring function is defined as

$$f_r(h,t) = \mathbf{h}^T diag(\mathbf{r}) \mathbf{t} \tag{2.3.2}$$

This score captures pairwise interactions between only the components of h and t along the same dimension, and reduces the number of parameters. However, this is over-simplified and can only deal with symmetric relations which is not powerful enough for general KGs.

2.3.3 Semantic Matching Model with Neural Networks

Apart from RESCAL and its extensions, another type of KG embedding technique is semantic matching model with neural networks. Semantic Matching Energy (SME) [11] conducts semantic matching using NN architectures. Given a fact (h,r,t), it first projects entities and relations to their vector embeddings in the input layer. The relation $\mathbf{r}$ is then combined with the head entity $\mathbf{h}$ to get $g_u(\mathbf{h},\mathbf{r})$, and with the tail entity $\mathbf{t}$ to get $g_v(\mathbf{t},\mathbf{r})$ in the hidden layer. The score function is finally defined as matching g_u and g_v by their dot product as

$$f_r(h,t) = g_u(\mathbf{h},\mathbf{r})^T g_v(\mathbf{t},\mathbf{r}) \tag{2.3.3}$$

Neural Tensor Network (NTN) [69] is another NN architecture. Given a fact (h,r,t), it first projects entities to their vector embeddings in the input layer. Then the two entities $\mathbf{h},\mathbf{t} \in \mathbb{R}^d$ are combined by a relation-specific tensor $\mathbf{M}_r \in \mathbb{R}^{d\times d\times k}$ (along with other parameters) and mapped to a non-linear hidden layer. Finally, the relation–specific linear output gives the score

$$f_r(h,t) = \mathbf{r}^T tanh(\mathbf{h}^T \mathbf{M}_r \mathbf{t} + \mathbf{M}_r{}^1 \mathbf{t} + \mathbf{M}_r{}^2 \mathbf{t} + \mathbf{b}_r \tag{2.3.4}$$

where $\mathbf{M}_r{}^1, \mathbf{M}_r{}^2 \in \mathbb{R}^{k\times d}$ and $\mathbf{b} \in \mathbb{R}^k$ are relation-specific weight matrices and bias vectors, respectively.

Multi-Layer Perceptron (MLP) [26] is a simpler approach where each relation (as well as entity) is associated with a single vector. Given a fact (h,r,t), the vector embeddings are concatenated in the input layer and mapped to a non-linear hidden layer. The score is then generated by a linear output layer as

$$f_r(h,t) = \mathbf{w}^T tanh(\mathbf{M}^1 \mathbf{h} + \mathbf{M}^2 \mathbf{r} + \mathbf{M}^3 \mathbf{t}). \tag{2.3.5}$$

where $\mathbf{M}^1, \mathbf{M}^2, \mathbf{M}^3 \in \mathbb{R}^{d\times d}$ are the first layer weights, and $\mathbf{w} \in \mathbb{R}^d$ is the second layer weights, all shared across different relations.

2.4 Improper Rating-range Problem and Rating Conversion

The improper rating-range problem occurs because each user has their own rating pattern; the ratings of each user are in different ranges. For example, in a 1–5 rating system, user *A* always rates items within a range of 1–3, while user *B* always rates items only in a 3–5 range. The meaning of a rating of 3 by user *A* is different from a rating of 3 by user *B*. From this scenario, A rating of 3 by user *A* would then be similar to a rating of 5 by user *B*. This means that using the actual rating from one user to recommend the item for the target user in the prediction step is improper and this may bring about low accuracy for the recommendations.

Therefore, some researchers have tried to adjust the ratings in different ranges to be in the same range. The first method is normalization [43, 44]. It converts ratings into a specific range, usually between 0 and 1 where 'most dislike' will be mapped to '0' and 'most like' will be mapped to '1'. Many normalization methods are proposed based on different assumptions, such as linear normalization, Gaussian normalization, and decoupling normalization.

2.4.1 Linear Normalization

This method maps ratings based on the maximum and minimum of personal user ratings. By using the linear function, the normalized rating value $r_{u_a}^{new}$ for the user u_a's specific rating is computed as:

$$r_{u_a}^{new} = \frac{r_{u_a}^{old} - r_{u_a,min} + 1}{r_{u_a,max} - r_{u_a,min} + 1}, \tag{2.4.1}$$

where $r_{u_a}^{old}$ is an original rating of u_a, $r_{u_a,max}$ and $r_{u_a,min}$ denote the maximum and minimum ratings user u_a has rated, respectively. This normalization method maps ratings based only on maximum and minimum of the personal user ratings.

2.4.2 Gaussian Normalization

This method considers two factors that affect the variance of ratings among users with similar interests [43]. The first factor is a difference of a rating from the average ratings. This factor relates to the fact that some users are more tolerant and tend to give higher ratings than others. Another factor is the difference in users' rating scales. This comes from the fact that some users tend to assign items to a narrow range of ratings, whereas other users tend to assign items to a wide range. Combining these two factors, the ratings of each user are subtracted from his average and divided by the variance of his ratings, as expressed by:

$$r_{u_a}^{new} = \frac{r_{u_a}^{old} - \bar{r}_{u_a}}{\sigma_{u_a}}, \tag{2.4.2}$$

where $\bar{r}_{u_a}$ and σ_{u_a} are an average and a standard deviation of user ratings, respectively.

2.4.3 Decoupling Normalization

This method converts a user rating on an item into a probability for that item to be favored by the user [44]. When the rating r_{u_a} is going to be normalized, the probability is determined based on two factors. First, a ratio between two numbers: the number of items which was rated no more than value r_{u_a} by the user u_a and the number of all items that the user u_a has rated. The high ratio means the rating r_{u_a} is likely to be favored by the user. The second factor is a ratio between the other two numbers: the number of items which was rated value r_{u_a} by the user u_a and the double number of all items that the user has rated. The low ratio means the rating r_{u_a} is likely to be favored by the user. Based on these two factors, a special formula; called halfway cumulative distribution was proposed as:

$$r_{u_a}^{new} = \frac{|\{v_j \in I_{u_a} | r_{a,j} \leq r_{u_a}^{old}\}|}{I_{u_a}} - \frac{|\{v_j \in I_{u_a} | r_{a,j} = r_{u_a}^{old}\}|}{2|I_{u_a}|}, \tag{2.4.3}$$

where I_{u_a} denotes the set of items to which user u_a has rated.

However, normalization is not enough because the normalized rating is converted on the basis of the data of the original user only. The problem can occur if two target users have different rating patterns but these two target users have the same set of friends. If friend's normalized ratings are used for prediction, the recommendation will be the same, which is improper. For example, target user *A* usually rated items with a score '0.3' (normalized), while target user *B* usually rated items with a score '0.7' (normalized). If these two users have the same set of friends and the predicted rating from this set of neighbors is '0.8', these two target users will have the same predicted rating of '0.8'. Although the predicted rating '0.8' does not have much effect on user *B*, it seems quite a high score to user *A* because his usual rating is'0.3'.

Thus, in order to overcome this problem, an algorithm that can converts ratings from one user to the perspective of the target user is needed to improve the accuracy level of the recommendations. We call this algorithm as *rating conversion*.

2.5 Deep Learning and Attention Mechanism

In recent years, NNs and deep learning have been applied to many research fields. Deep learning is now becoming the most popular technique for information retrieval and RSs [81].

Matrix factorization (MF) [49] is a standard baseline technique for CF that models the relationship between user and item via their inner product, with several implementations of deep learning models now being based on MF [30, 37]. One such approach is neural MF (NeuMF) [37], a recent state-of-the-art deep learning model that combines MF with a multilayer perceptron (MLP).

The *attention mechanism* is widely used and is successful for learning a weighted representation across multiple samples. It was first introduced by Bahdanau, D., et al. in [5]. Inspired by human visual attention, the goal of the attention mechanism is to reduce noise and select more informative features for the final prediction. The attention mechanism has recently become popular in the fields of computer vision, natural language processing, and RSs [16, 17, 52, 70]. In attention-based methods, inputs are weighted with attention scores. As such, calculating the attention scores is the most important part of neural attention models.

Despite many advanced models emerging each year, with several types of NN and deep-learning techniques being applied to RSs, RSs still suffer from several challenges, such as data sparsity and cold start problems. Therefore, applying only deep learning to RSs might not be the best solution, so many researchers have tried to apply deep learning with other techniques or additional information to RSs. This thesis also adopts the attention mechanism in the model and combines it with the neighborhood-based CF concept in building a hybrid CF RSs.

Chapter 3

Related Work

3.1 Existing Recommender Systems

3.1.1 Overview

Due to the problem of information overload, many RSs have been developed over decades, to suggest and recommend personalized items to individual users [1, 8, 66]. As mentioned in Chapter 2.1, traditional RSs can be categorized in three main approaches which are the CF approach, the content-based approach, and the hybrid approach. The most popular and widely used approach in RSs is CF, and my thesis focuses on hybrid CF. Therefore, in this section, I will focus on the existing CF RSs.

Over the past few years, a large number of CF RSs have been developed to solve several problems, e.g., cold-start problem, sparsity problem, scalability, explainability, etc. [57] In this thesis, the proposed model is proposed to solve the sparsity problem, explainability, and the improper rating-range problem.

Sparsity problem occurs when the dataset contains very few ratings from users. Due to the sparseness of ratings in the user–rating matrix, the model cannot generate a satisfying recommendation for individual users. This problem is the problem of CF RSs, which gives unique opportunities to new researchers to find new ways to predict the missing ratings.

According to existing CF RSs, the model-based CF approach becomes more popular than the neighborhood-based CF because of its advantages. However, the model-based CF approach still has flaws. Although features of neighborhood are embedded implicitly in the model-based CF, the inference is intractable because of the hidden/latent factors. Therefore, in this thesis, I argue that the existing model-based CF RSs are not sufficient to yield satisfactory embeddings for CF. I believe that the concept of neighbors is still important for RSs. In this thesis, I would like to integrate the concept of neighbors directly into the model-based CF

and propose a hybrid CF model. The purpose of integrating neighbors directly into the model is to increase the explainability of the model. Instead of only hidden/latent factors which lack explainability, it will be clearer to explain how the recommendations come from when the neighbors are directly used in the model.

In this chapter, some of the existing CF RSs are briefly explained.

3.1.2 Collaborative Filtering Recommender Systems

Several CF RSs have been proposed over decades. Chen, R. et al. proposed a survey of CF RSs from traditional methods to hybrid methods based on social networks [19]. They provide a comprehensive review of existing CF RSs which summarize research papers related to CF RSs that were published before 2018.

As mentioned in 2.4.3 that CF RSs can be categorized into the neighborhood-based approach and the model-based approach, so in this subsection, I will show some of the existing neighborhood-based CF RSs and model-based CF RSs. For the hybrid CF RSs, I will explain and show some existing hybrid CF RSs in the next subsection.

Neighborhood-based Collaborative Filtering

Neighborhood-based CF recommendation algorithm obtains the similar relationship between users or items according to the user-item rating matrix, then recommends the items that are highly rated by similar users for the target user. The traditional neighborhood-based CF can be categorized into two types: the user-based CF and the item-based CF.

Typical user-based CF (resp. item-based CF) usually calculates the similarity between two users (resp. two items), and produces a prediction for the user by taking the weighted average of all ratings. Similarity calculation between users or items is an important part of this approach. Multiple measures, such as Pearson correlation (Eq. 2.1.2) and vector cosine based similarity are used for this approach. The cosine similarity between user u_a and u_b can be calculated by this equation:

$$sim(u_a, u_b) = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}. \tag{3.1.1}$$

For recommendation, top-n recommendation is the application RSs to produce a set of n items which might be satisfied by the target user.

Model-based Collaborative Filtering

The model-based CF RSs are developed using different data mining, machine learning algorithms to predict the missing rating. There are many model-based CF RSs that are proposed over decades. This approach requires a learning phase in advance for finding out the optimal model parameters before generating recommendations for users. Among the model-based CF RSs, the latent factor model (LFM), which is the approach that factorizes the user-item rating matrix into two low-rank matrices: the user feature and the item feature matrices. This LFM is very competitive and widely adopted to implement RSs. Singular Value Decomposition (SVD) [48] and Matrix Factorization (MF) [49] are the best-known methods used in the model-based CF. Both SVD and MF all take advantage of LFM.

In recent years, deep learning is applied to model-based CF RSs. Any neural differentiable architecture is considered to be 'deep learning' if it optimizes a differentiable objective function using a variant of Stochastic Gradient Descent (SGD). There are many different architectures in deep learning, such as the Multilayer Perceptron (MLP), the Autoencoder (AE), the Attentional models (AM), etc. These architectures are utilized in RSs to build deep learning–based RSs. Neural Collaborative Filtering (NCF) [37] is one of the most popular state-of-the-art unified frameworks combining MF with the MLP.

3.1.3 Hybrid Collaborative Filtering Recommender Systems

The term hybrid CF RSs can refer to three different kinds of RSs which are:

- RSs that combine the neighborhood-based CF and the model-based CF approaches
- RSs that combine the user-based and item-based neighborhood-based CF approaches
- RSs that combine the CF approach (i.e, neighborhood-based or model-based) with another approach.

In this thesis, my proposed model is categorized in the first one, which is the hybrid CF RS that combines the neighborhood-based CF and the model-based CF approaches together.

A hybrid CF model with a deep structure for RSs [27] is proposed by Dong, X. et al. This hybrid CF model integrates deep representation learning and MF together. They proposed a deep learning model called additional stacked denoising autoencoder (aSDEA) to integrate the additional information into the input. This approach seems similar to my proposed model in which they try to integrate side information of users and items as input of the model. They use additional information about neighbors explicitly. However, the procedure of utilizing neighbors is different. Moreover, they binarize explicit data by keeping the ratings of four or

higher and interpret them as implicit feedback, while I try to keep and make the most use of all information.

Recently, Pirasteh, P. et al. proposed an enhanced hybrid CF that utilizes movie genres as additional information to calculate the similarity between items and users [60]. They calculate four different similarities based on three matrices, namely: user similarity based on ratings, item similarity based on rating, user similarity based on genres and item similarity based on genres. Then compute a final prediction for a user (or an item) based on a linear combination of the individual estimated scores using an ordinary least-squares regression. This hybrid CF model is straightforward and easy to understand. However, it still relies on the neighborhood–based approach and it's not an end-to-end hybrid CF model.

In [45], Juan, W. et al. suggested a hybrid CF RS that combines the K nearest neighbor (KNN) [23] model and the Extreme Gradient Boosting (XGBoost) [20] model and leverages the scores predicted by the model-based personalised recommendation algorithm as features to overcome data sparsity and cold start problem in RSs. The algorithmic principle behind XGBoost is to pick a few samples and features to construct a classification model.

3.2 Knowledge Graph Embedding based Recommender Systems

KG embedding involves embedding the components of KG such as entities and relations into a continuous vector space. Many KG embedding techniques have recently been proposed and are rapidly gaining attention [76].

Nowadays, KG embedding techniques have been adopted in various fields, including RSs. Researchers in RSs field have applied KG embedding techniques to RSs for which the RSs problem is defined as link prediction in a KG comprising users and items as the entities and ratings as the relations [34]. For example, user u_1 rated item i_1 can be transform into triple in KG setting as $(u_1, rated, i_1)$.

These KG embedding techniques are being applied to RSs to mitigate the challenges such as data sparsity and cold start problems. Several approaches integrate *side information* into the RSs and treat the resultant data as a KG, aiming to address these issues. In this thesis, I show some related KG embedding-based RSs. They can be roughly categorized into translation-based embedding models in RSs and KG-based NN RSs.

3.2.1 Translation-based Embedding Model in Recommender Systems

Inspired by the translation-based embedding model, several works applied the translation-based embedding model in RSs.

Translational Models for Item Recommendation

Palumbo, E. et al. proposed translational models for item recommendation [59]. They showed that the item recommendation can be seen as a specific case of KG completion problem, where a special property called 'feedback' has to be predicted. See Figure 3.1. Feedback can be seen as a relation in KG. They applied TransE [12], TransH [78], and TransR [53] to map entities and relations into a vector space and generate KG embeddings. Then, they tried to complete the KG by predicting the feedback between a pair user and item. Finally, they generated the item recommendation by ranking a set of items based on the predicted feedback.

Learning over Knowledge-Base Embeddings for Recommendation

Zhang, Y. proposed another translational embedding based method. [82] They proposed to learn over heterogeneous knowledge base embeddings for personalized recommendation by constructing the user-item knowledge graph, then learn the knowledge base embeddings with the heterogeneous relations collectively and use these embeddings to produce personalized recommendations. In their work, they defined 5 types of entities and 6 types of relations, where the entities include *user, item, word, brand*, and *category*, while the relations include *buy*,*belong to category*, *belong to brand*, *mention word*, *also bought*, and *also view*. Figure 3.2 shows an example of the constructed user-item KG.

Constrained Preference Embedding for Item Recommendation

Wang et al. proposed a method called constrained preference embedding (CPE) to model users' behavior. [77] They adopt the idea from translation-based embedding model to embed user's behavior information in a high-dimensional space together with users and items. In this context, users and items are entities and user's behavior (feedback from user to item) are relation. The users, items, and relations are represented by d dimensional vectors.

They proposed two item recommendation algorithms CPE-s and CPE-ps based on their CPE assumption. CPE-s adopt the idea from TransE [12] and CPE-ps adopt the idea from TransH. [78]

Although all of [59, 82, 77] try to represent users, items, and relations in embedding vector form, their representation may be improper and inaccurate. The problem can occur if

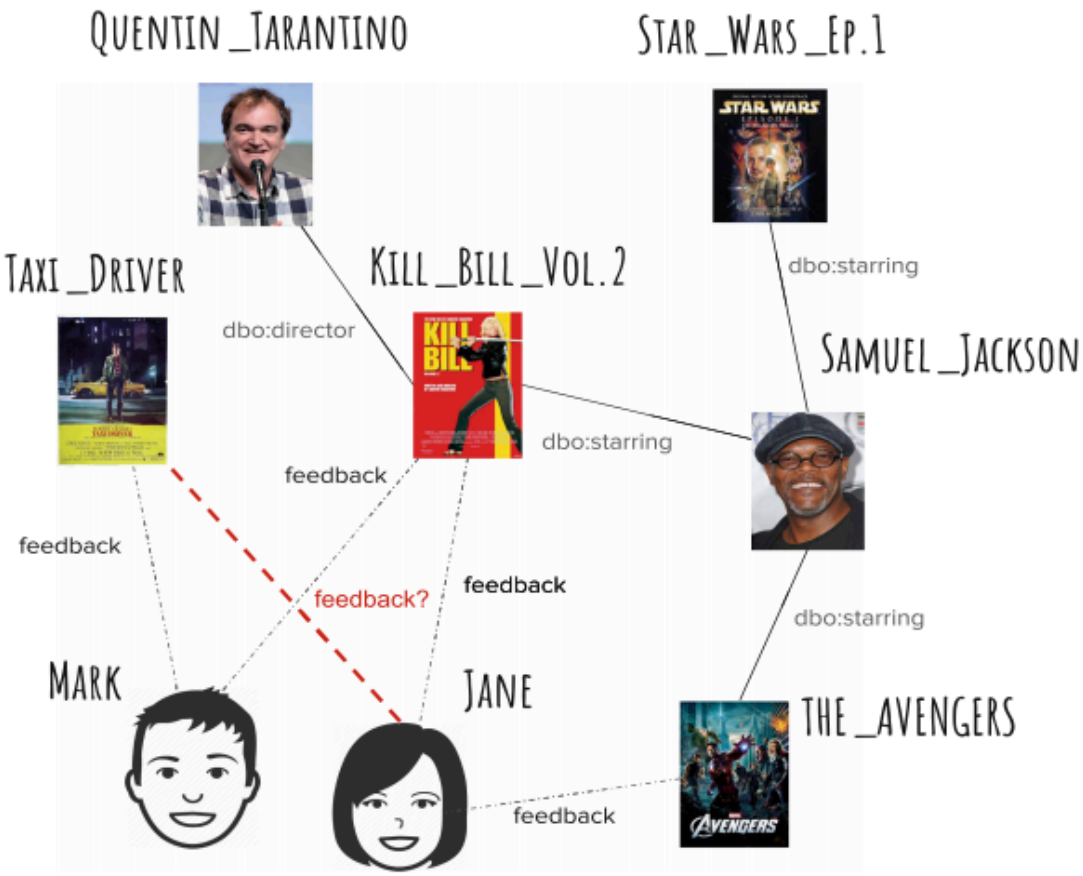

Figure 3.1: Recommending items as a knowledge graph completion problem. Adapted from [59].

the rating pattern of each user are not in the same range. These works do not consider this problem. This means using the obtained rating vector from these models in the prediction step may lead to inaccurate recommendations.

3.2.2 Knowledge Graph-based Neural Network Recommender Systems

Apart from applying the translation-based embedding model to RSs, some researchers have applied KG embedding approach to NN RSs model, which can be called KG-based NN RSs.

Latent Relational Metric Learning

Latent Relational Metric Learning (LRML) [70] is an attention-based memory-augmented neural architecture that uses a latent relation vector to model the relationship between user–item pairs in metric space, proposed by Tay et al. LRML tries to mitigate the problems with

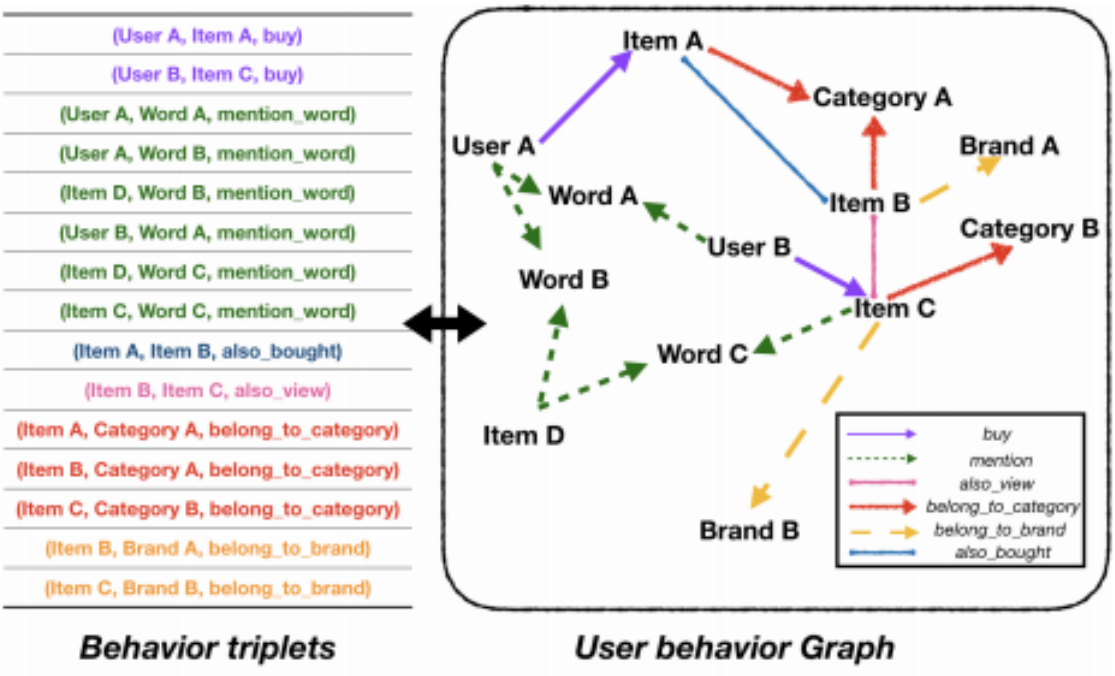

Figure 3.2: An example of user-item KG. Adapted from [82].

CML [39], whose scoring function is geometrically restrictive because the objective function of CML aims to minimize the distance between each user–item pair in Euclidean space. The scoring function is defined as:

$$s(p,q) \approx \|p-q\|_2{}^2, \tag{3.2.1}$$

where p, q are the user and item vectors respectively.

With this objective function of CML, it causes geometric restrictive because the objective function tries to fit each user–item pair into the same point in vector space. This geometric inflexibility causes poor repercussions when the dataset is large or dense since CML tries to force all of a user's item interactions onto the same point.

Therefore, LRML adopts the scoring function from TransE [12] to model the latent relation vector of a user–item pair, instead of assigning them to the same spot, as does CML. Figure 3.4 shows the difference between LRML and CML.

LRML aims to model user and item pairs using relation vectors based on translational principle, i.e., $p + r \approx q$. The relation vector r is what separates LRML from simple metric learning approaches like CML. LRML's architecture is described in Figure 3.5. A simple high-level overview of LRML is described as follows.

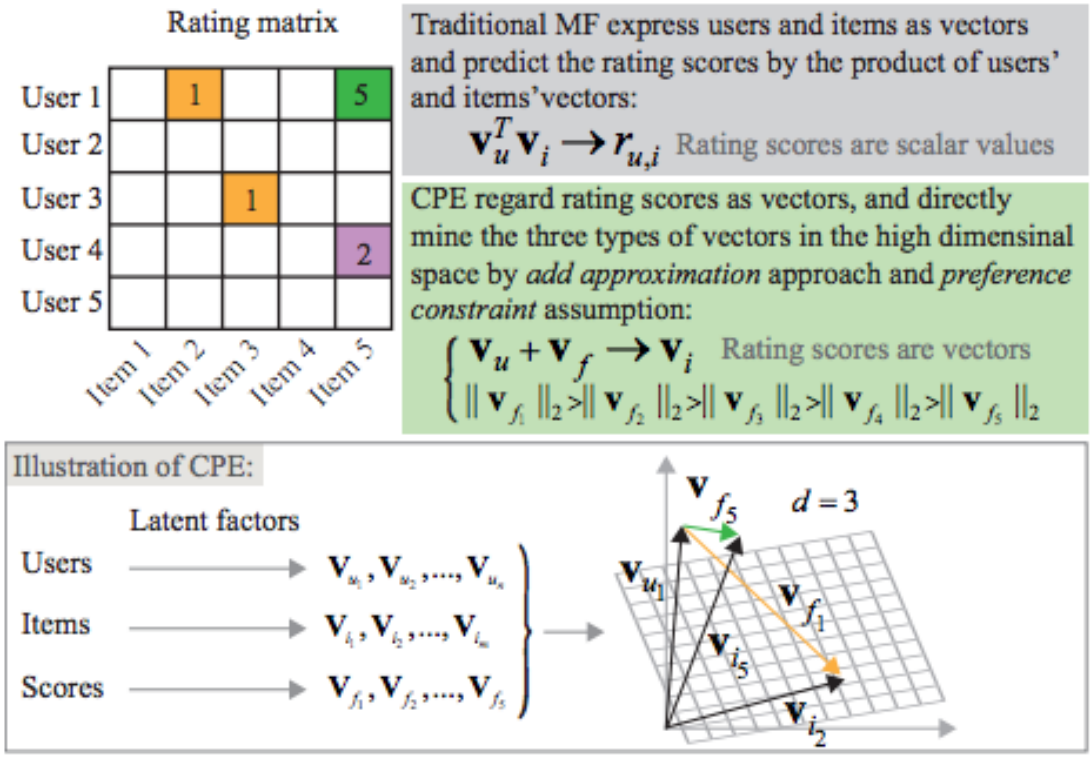

Figure 3.3: Illustration of CPE. Adapted from [77].

1. Users and items are converted to dense vector representations using an embedding layer. p and q are the user and item vectors respectively.

2. Given p and q a relation vector r is generated using a neural attention mechanism over an augmented memory matrix $\mathbf{M}$. The relation vector r is dependent on user and item, and is learned to explain the relationship between user and item.

3. LRML optimizes for $\|p+r-q\| \approx 0$ using pairwise hinge loss and negative sampling like in TransE.

Because TransE [12], as proposed by Bordes et al., is both the simplest and the most representative translation-based KG embedding technique, it is widely used in various fields. It represents both entities and relations as vectors in the same vector space. This thesis is inspired by the ideas of LRML and TransE to enable learning an appropriate representation of users, items, and latent relations. However, my proposed model in this thesis obtains the latent relation vector differently from LRML which will be described in detail later.

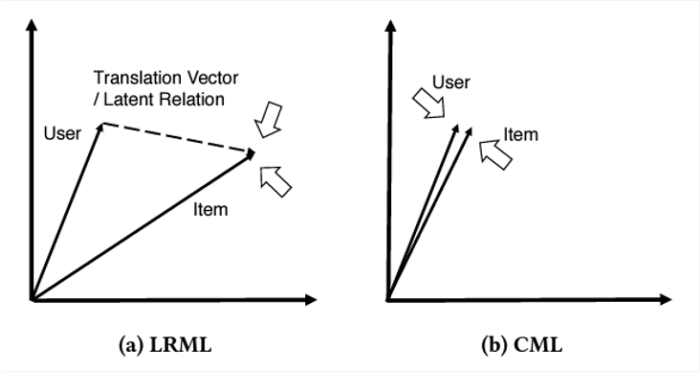

Figure 3.4: Geometric Comparisons of LRML and CML. Adapted from [70].

3.3 Rating Conversion Techniques

In attempts to overcome the problem aforementioned, many rating conversion techniques have been proposed over the years. Those techniques include linear mapping, Lathia's transpose function, Warat's transpose function, Ayub's rating conversion and rating conversion on multi-criteria RSs. In addition, I also proposed a rating conversion technique that works specifically for trust-based RSs.

3.3.1 Linear Mapping

Linear mapping is similar to linear normalization. [2] It uses a linear function to map the ratings, but the different point is Linear mapping maps the original rating range to the target user's rating range instead of range [0,1]. The linear mapping function that will map user u's rating to user A' rating range is defined as follows:

$$Linear_{u \to A}(R) = (R_u - r_{u,min}) \cdot \frac{r_{A,max} - r_{A,min}}{r_{u,max} - r_{u,min}} + r_{A,min} \tag{3.3.1}$$

where $r_{A,max}$ and $r_{A,min}$ denote the maximum rating and minimum rating that user A has rated, respectively.

Although the linear mapping approach can solve some problems of normalization, some still occur. First, if there are users who have different rating patterns but have the same maximum rating and minimum rating, the predicted rating from the same friends is still the same as occurred in the normalization. Moreover, if the target user has rated items by only one rating value, all ratings from all friends will be mapped to that value. This makes an improper prediction since all items receive the same predicted rating.

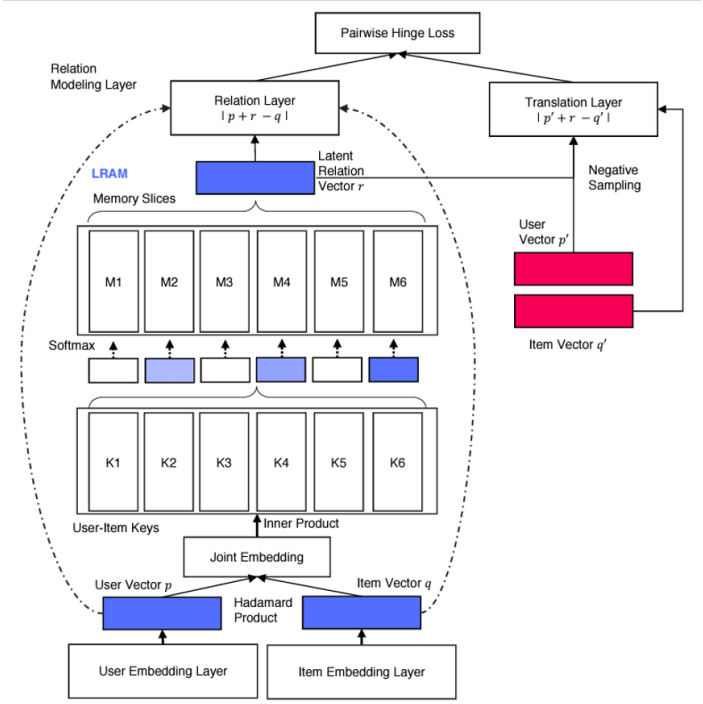

Figure 3.5: Illustration of LRML architecture. Adapted from [70].

3.3.2 Lathia's Rating Conversion Method

Lathia's rating conversion method is proposed by Lathia, N. et al. [50] This method converts the ratings between a pair of users based on co-rated items, which is the set of items that have been rated by both users of that pair. For example, if user A is the target user and we would like to convert user u's rating into user A's perspective. Consider a set S of co-rated items rated by both user A and user u. If user A has rated for almost all items in S greater than user u has rated, Lathia's method assumes that user u's original rating will be converted to greater value in user A's aspect. The converted rating is computed by a weighted mean between the quantity of each group of ratings in S. Let $lower_R$, $same_R$ and $higher_R$ denote the groups of items that a target user A has rated less, equal and more than the original rating R of user u, respectively. The $lower_R$ group decreases the rating from R to $(R-1)$, $same_R$ unchanges R and $higher_R$ group increase the rating to $R+1$. Lathia's method can be described by the following.

$$Lathia_{u\to A}(R) = \frac{(R-1)|lower_R| + R|same_R| + (R+1)|higher_R|}{|lower_R| + |same_R| + |higher_R|} \tag{3.3.2}$$

However, the problem still occurs if this method is used. For example, suppose user u have rated fourteen co-rated items with rating '4', and target user A have rated less than

'4' on five items of them, equal to '4' on three items of them, and greater than '4' on the remaining six items of them. Computed by Lathia's method, it will convert rating '4' of user u to '4.07' of user A's perspective. As for the rating that has no historical experience, the converted rating remains the same as the original one.

3.3.3 Warat's Rating Conversion Method

Warat's rating conversion method is proposed by Warat et al. [15]. This work was proposed to overcome the insufficient co-rated items problem and the rating shifting problem. It was designed to deal with rating in scalar form. Warat's method consists of two components: the original value and the adjusting term. The original value is the rating to be converted and the adjusting term is an average of the difference between the original rating and the related target ratings (ratings of the target user on items corresponding to the original rating). Due to the sparsity of data, there is a chance that there are no actual related target ratings. In that case, the pseudo ratings derived from matrix factorization [49] are used instead.

The Warat's function controls the adjusting term with the reducing term which are confident of generated pseudo ratings and distribution of ratings. The less error and less deviation of the considered rating lead to more suitable converted ratings. The transposed rating can be calculated as:

$$W_{u\to A}(R) = R + \frac{\sum_{i\in\beta_{uR}}(r_{Ai}-R)}{|\beta_{uR}|}\cdot Dist_{Au}\cdot Conf_A. \tag{3.3.3}$$

For more understanding, we show how Warat's method works in three cases. For the first case, if user u and user A_1 shared the same set of items with ratings (2, 2, 2) and (4, 4, 4), respectively. Since their ratings contain no distribution, if the pseudo ratings from user A have no error, '2' of user u will be converted exactly to '4' of user A's perspective. In the second case, there exists another user A_2 who rated the same set of items with distributed ratings (3, 4, 5), no error. Although the average rating is '4', the converted rating from user u is only '3.1' since the distribution term reduced the adjusting term from '+2' to '+1.1'. The last case, if there is an error on user A_2's derived pseudo ratings with confidence of 0.8, the adjusting term is reduced to '0.6', and the converted rating from user u is '2.6'.

However, most of them were proposed to deal with ratings only in scalar form. Our proposed method tries to deal with ratings into vector form, so in this work we propose a new technique that can convert the ratings in vector form. The details of the proposed method are described in the next section.

3.3.4 Ayub's Rating Conversion

Ayub, M. et al. proposed a Modeling user rating preference behavior to improve the performance of the CF based RSs [4]. In this work, they tried to solve the improper rating-range problem too. They claim that the standard Pearson correlation coefficient (PCC), which is commonly used for similarity measurement between users in CF RSs, ignores different rating pattern of each user. Therefore, they proposed an improved similarity measure method that uses the user's rating preference behavior (RPB) pattern to find similar users. This proposed similarity measure is named as improved PCC weighted with RPB (IPWR).

First, they proposed a function RPB between user a and b, $RPB_{a,b}$ in two ways: using variance and using standard deviation (SD). The cosine function is used to model the RPB of two users. It can be calculated by follows:

$$RPB_{(a,b)} using\ var = \cos(|\bar{R}_a - \bar{R}_b| \cdot |var_a - var_b|), \tag{3.3.4}$$

$$RPB_{(a,b)} using\ SD = \cos(|\bar{R}_a - \bar{R}_b| \cdot |SD_a - SD_b|). \tag{3.3.5}$$

Then they improve the standard PCC because the standard PCC ignores users' rating pattern. The improved PCC is denoted by Sim_IPCC. It can be calculated as follows:

$$Sim\_IPCC_{(a,b)} = \frac{\sum_{j \in I_a \cap I_b} \{(R_{a,j} \cdot \bar{R}_a) - (R_{a,j} \cdot \bar{R}_j)\} * \{(R_{b,j} \cdot \bar{R}_b) - (R_{b,j} \cdot \bar{R}_j)\}}{\sqrt{\sum_{j \in I_a} \{R_{a,j} \cdot \bar{R}_a) - (R_{a,j} \cdot \bar{R}_j)\}^2} \sqrt{\sum_{j \in I_b} \{R_{b,j} \cdot \bar{R}_b) - (R_{b,j} \cdot \bar{R}_j)\}^2}}. \tag{3.3.6}$$

The final IPWR similarity measure is a weighted combination between PCC and RBP which can be calculated as follows:

$$IPWR_{(a,b)} with\ variance = \alpha \cdot RPB_{(a,b)} using\ var + \beta \cdot Sim\_IPCC_{(a,b)}, \tag{3.3.7}$$

$$IPWR_{(a,b)} with\ SD = \alpha \cdot RPB_{(a,b)} using\ SD + \beta \cdot Sim\_IPCC_{(a,b)}. \tag{3.3.8}$$

The final recommendation are generated as follows:

$$\hat{R}_{a,i} = \bar{R}_a + \frac{\sum_{b \in NN} IPWR_{(a,b)} \cdot (R_{b,i} - \bar{R}_b)}{\sum_{b \in NN} |IPWR_{(a,b)}|}, \tag{3.3.9}$$

where b denotes a user belonging to the nearest neighbor (NN) network of the user a.

3.3.5 Rating Conversion on Multi-criteria RSs

Apart from rating conversion on a single criterion neighborhood-based CF mentioned above, some have proposed rating conversion methods on multi-criteria RSs. In some systems, the users can express their preferences in multiple aspects of items in addition to an overall rating. The multi-criteria system grants a better opportunity to analyze the user preferences in more detail, and leads to more personalized and effective recommendations.

Sitkrongwong, P. et al. proposed a multi-criteria rating conversion without relation loss for RSs [68]. They proposed a novel method that simultaneously converts all criteria ratings between users to maintain their implicit relations. The multi-criteria ratings are first normalized by variances of users and principle component analysis (PCA) is applied to extract user preference patterns. Such patterns are then used for multi–criteria rating conversion.

After the multi-criteria rating patterns of all users are extracted, they can be used to convert the rater multi-criteria ratings on each item into the active user's aspect.

Suppose the task is to convert rater u_a multi-criteria ratings on target item i into an active user u_b's aspect. First, the rating vector $\mathbf{r}_{aj}$ of item v_j is selected from $R^V_{u_a}$. This vector is then subtracted by its mean value to create the *modified rating vector* $\mathbf{r}^M_{aj}$. To convert to the aspect of u_b, $\mathbf{r}^M_{aj}$ is projected into the same plane of u_b by multiplying with the feature matrix $\mathbf{Q}_{u_b}$ of u_b by the following:

$$\sigma_{bj(a)} = \mathbf{Q}_{u_b}(\mathbf{r}^M_{aj})^T. \tag{3.3.10}$$

Before it can be used further, $\sigma_{bj(a)}$ needs to be turned back into the original rating scale by the following:

$$\theta_{bj(a)} = \sigma_{bj(a)} + \mu_{u_a}, \tag{3.3.11}$$

where $\mu_{u_a} = [\mu_{u_a,1} \cdots \mu_{u_a,K}]$ is a mean rating vector which each element contains the mean value of ratings in each criterion of user u_a. After that, the converted rating is used in the rating prediction step.

Chapter 4

Utilizing Neighbors in a Hybrid CF Model

4.1 Introduction

In recent years, as the amount of information created by people's daily activities has increased, users have found it challenging to select products that best suit them from among the many options available. As a result, filtering tools that can find and recommend appropriate and relevant items to users are important. Recommender Systems (RSs) have evolved through time and have become a key approach for dealing with information overload issues.

In RSs, there are many approaches. The most prevalent and popular approach for recommending items to the *target user* (the user to whom recommendations are targeted) is the collaborative filtering (CF) approach, which is based on the similarity of users or items to previous interactions. It utilizes historical interaction (e.g., clicks, rates, purchases, etc.) to infer the user's preference and recommend items based on the matching score between the target user and the target item.

Typically, CF can be further divided into two approaches: neighborhood-based approach and model-based approach. Although the neighborhood-based approach is simple and works reasonably well in practice, it requires a high cost in terms of computing time and spatial complexity. Also, it can suffer from sparsity problems. Therefore, nowadays, model-based approach has gained more popularity than neighborhood-based approach because it is more capable of handling the problem of sparsity and scalability. However, model-based approach requires a great resource to develop the model and may lose information when using dimensionality reduction. Moreover, model-based approach still suffers from interpretability.

From the previous paragraph, it indicates that both neighborhood-based approach and model-based approach provide both advantages and disadvantages. Whether neighborhood-based approach or model-based approach, they have their own characteristics and can provide accurate results depending on the given task. It would be great if we can combine their advantages in building a hybrid CF model for recommendation which leverage the advantages of both neighborhood-based and model-based CF approaches while weakening their disadvantages. The challenges of this idea is how to build and propose the hybrid CF model which maintains all of the advantages of both CF approaches and reduces their disadvantages. Therefore, this chapter will focus on how to combine both types of CF into the one single hybrid CF model.

Recently, most of the research in the RSs field often focuses on the model-based CF approach which is now on-trend rather than the old-fashion neighborhood-based CF approach. I also agree that the model-based CF approach is a good choice for building recommender systems because of its advantages. However, many researchers still doubt in model-based CF's prediction because of the lack of explainability. It is difficult to explain how is that recommendation come from. It would be great if the results can be explained easily like the result from the neighborhood-based CF approach.

I strongly believe in the idea of traditional neighborhood-based CF approach because of its simplicity and explainability, so this thesis proposes a hybrid CF which incorporates the idea of neighborhood-based CF into model-based CF model. When applying the neighborhood-based idea into model-based CF, we should carefully consider how to include neighbors into model while preserving characteristics of both neighborhood-based CF and model-based CF in this proposed hybrid CF model.

Moreover, to make the model more explainable, explicit feedback should be used in this hybrid CF model because explicit feedback can express more characteristics and can help the model to learn better user and item representations, which will further result in better recommendations. Thus, the main idea of this chapter is to incorporate the neighborhood-based CF idea into a model-based CF which focuses on how to incorporate explicit feedback into an end-to-end hybrid CF model.

In this chapter, there are two approaches to utilizing neighbors into the hybrid CF models, which are 1) simple approach to utilize neighbors in TransRS [73], and 2) utilizing neighbors in AHCF [74] which is the end-to-end hybrid CF model.

4.2 Simple Approach to Utilize Neighbors in Hybrid CF

In this thesis, hybrid CF refers to a combination between neighborhood-based CF and model-based CF, which can be roughly categorized into two types: building blocks model and end-to-end model. The building block model can be separated into several blocks and each block can do its task within itself, while end-to-end model refers to training a possibly complex learning system represented by a single model (specifically a deep neural network) that represents the complete target system, bypassing the intermediate layers usually present in traditional pipeline designs.

This subsection focuses on utilizing neighbors in a building blocks hybrid CF model. This proposed method of utilizing neighbors in the hybrid CF model is proposed in TransRS [73], which is a simple approach to integrate the concept of neighborhood into hybrid CF model.

Note that TransRS [73] is not an end-to-end hybrid CF model, but consists of three main steps for the recommendations. In this work, I applied model-based CF into neighborhood-based CF, so the main part is on the neighborhood-based CF side. This proposed model consists of three main steps which are: 1) Embedding users and items into the same feature space, 2) Rating Conversion and 3) Rating Prediction. As a building blocks model, the output of the previous step will be the input of the next step.

The next subsection will describe the details of how to utilize neighbors in the end-to-end hybrid CF model.

4.3 Utilizing Neighbors in End-to-end Hybrid CF Model

In the previous subsection, I have described a simple approach to utilized neighbors in TransRS [73], which is the building blocks hybrid CF model. This subsection demonstrates how to utilize neighbors in the proposed end-to-end hybrid CF model, AHCF [74], which the utilization of neighbor is more complex than the previous one.

Since the main idea of this subsection is how to apply neighborhood-based CF concept into model-based CF model, the model structure, including user and item representations should be carefully considered. This hybrid CF model cannot rely on traditional neighborhood-based or model-based CF model, so in this chapter, I propose **a novel user and item representations** instead of using traditional embeddings.

Most traditional RSs always represent users and items in the form of an embedding vector. They usually represent each user or item by a single embedding vector. However, those embedding vectors cannot store specific explicit feedback (i.e., rating score) within

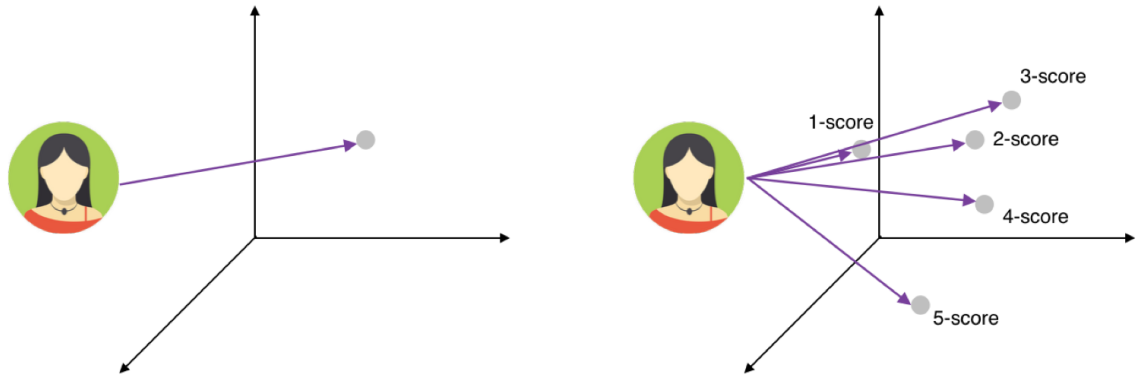

Figure 4.1: An illustration of (A) traditional user embedding and (B) The proposed user embedding (UR-matrix)

themselves. Even though some work use explicit feedback in their model-based CF model, they only use ratings as a label for optimization [3, 21, 14]. Those ratings are not directly incorporated as input and ratings do not play an important role in their model. Also, traditional user and item embeddings do not indicate specific characteristics of user's ratings or item's ratings.

Since my main intention is to apply the neighborhood-based CF concept to model-based models, explicit feedback, which is the common form of input of neighborhood-based CF models is also an important point to consider. Therefore, I would like to incorporate explicit feedback (i.e., ratings) directly into this proposed model. In order to do that, traditional user and item representations (i.e., single embedding vector) are unsuitable for my purpose because a single vector could not store the individual rating patterns and characteristics of each user and item. Therefore, instead of using a single embedding vector, I propose to use a dense matrix to represent each user and item, called the user representation matrix (*UR-matrix*) and the item representation matrix (*IR-matrix*), respectively. Note that these user and item representations are then available for rating conversion in Chapter 5.4.4.

UR-matrix and IR-matrix are proposed to store each user's and item's rating information, in their matrix representations respectively. Unlike traditional single embeddings which can represent each user and item without specific rating information (or it can be called implicit feedback), these proposed UR-matrix and IR-matrix can store and represent users and items with more characteristics by specific rating scores. Refer to Figure 4.1, this illustration shows differences between traditional user embedding and the proposed UR-matrix. In traditional RSs models, one user's embedding is usually represented by a single embedding vector and

it will be mapped to a specific point in vector space, showing the overall characteristics of the user and item. On the other hand, my proposed user's UR-matrix (resp. item's IR-matrix) is represented by a dense matrix with s rows, so it will be mapped to s points in the vector space which can capture and specify more characteristics of the user than the traditional way. According to Figure 4.1, suppose that this system accepts rating scores of range 1–5, so the UR-matrix of user and IR-matrix of item will be mapped to 5 different points in the vector space. Each point represents how the user rate item with specific s score and how item is rated by users with specific s score. These matrices will then be used for rating conversion purposes, which are described in Chapter 5.4.4.

4.3.1 User Representation Matrix (UR-matrix)

For a user u, the UR-matrix is denoted as $\mathbf{U}_u \in \mathbb{R}^{z \times d}$, where d is the dimensionality of user and item embeddings and z is number of rating scores. The s-th row of $\mathbf{U}_u$ corresponds to a rating score s and represents an embedding vector of u for s (e.g., the first row of the matrix represents score 1).

4.3.2 Item Representation Matrix (IR-matrix)

Similar to UR-matrix, for an item v, the IR-matrix is denoted by $\mathbf{V}_v \in \mathbb{R}^{z \times d}$, where d is the dimensionality of user and item embeddings and z is number of rating scores. The s-th row of $\mathbf{V}_v$ corresponds to a rating score s and represents an embedding vector of v for s (e.g., the first row of the matrix represents score 1).

Suppose that the RS accepts rating scores of $\{1,2,3,4,5\}$, then the UR-matrix and IR-matrix comprise five rows and each row represents each rating score in $\{1,2,3,4,5\}$. In this way, the UR-matrix represents how each user rates items and the IR-matrix represents how items have been rated by users.

Finally, $\mathbf{U} \in \mathbb{R}^{m \times z \times d}$ and $\mathbf{V} \in \mathbb{R}^{n \times z \times d}$ are embedding tensors that respectively store user and item embeddings, where m and n are the total numbers of users and items, respectively.

4.4 Discussion

In the previous two subsections, the details of how to utilize neighbors in TransRS [73] and AHCF [74] are described in detail.

Although both TransRS [73] and AHCF [74] are the hybrid CF model which combine neighborhood-based CF approach and model-based CF approach together, the structures of their models are completely different. In TransRS [73], it can be considered as 'how

to apply model-based idea into neighborhood-based CF model', while in AHCF [74], it can be considered as 'how to apply neighborhood-based idea into model-based CF model'. Normally, the neighborhood-based CF model is usually easy to implement. It requires a set of similar users based on cosine similarity or Pearson's correlation as an input and then takes the weighted average of their ratings to generate the prediction. As a set of similar users is a key idea of neighborhood-based CF model and it does not learn any parameter using optimization algorithm, so if we want to apply model-based idea into neighborhood-based CF model, the model-based idea can be applied in the step of 'finding a set of similar users' or 'generate the prediction'. In AHCF [73], the model-based idea is applied to both steps.

On the other hand, the model-based CF model uses machine learning to find user ratings of unrated items (e.g., PCA, SVD, NN, MF, etc.) and generate the prediction based on obtained ratings. The key difference of model-based approach from the neighborhood-based approach is that it does not require a set of similar users to generate the prediction. Therefore, if we want to apply the neighborhood-based idea into the model-based CF model, it can not be directly applied to model-based CF model without modification. The point is that the neighborhood-based idea should be implemented in a way of the model-based approach. Thereby, the procedures of how to utilize neighbors in TransRS [73] and AHCF [74] are totally different. It depends on the task and purpose.

Thereby, the procedures of how to utilize neighbors in TransRS [73] and AHCF [74] are totally different. It depends on the task and purpose. In my opinion, as the end-to-end learning has become popular for deep-learning applications in this era, the method of [74] described in Chapter 4.3.2 is more practical than [73] method described in Chapter 4.2.

Chapter 5

Neighbors Selection and Rating Conversion in Hybrid CF model

5.1 Introduction

According to the motivation in Chapter 1, this thesis intends to build a hybrid CF model which is a combination between neighborhood-based CF and model-based CF some problems to solve in both neighborhood-based CF and model-based CF. This chapter describes and discusses how to solve problems in neighborhood-based CF.

In the neighborhood-based CF model, typically, to predict the rating score of the target user toward the target item, CF RS aims to identify the set of the target user's friends and use their actual rating scores to determine how much the target user would like the target item. There are several approaches to finding a set of friends. If there is no explicit friendship relation in the dataset, most RSs often aim to find the set of *raters*, who have rated the target item in the past. In order to avoid any confusion in this work, I define the set of *friends* as a set of users who have rated the target item in the past. Then, the predicted rating score for the target user toward the target item is then calculated using the actual scores of friends. However, utilizing the actual rating scores from friends without conversion often leads to low-accuracy predictions because of the *improper rating-range problem* [15].

The improper rating-range problem occurs when the range of rating patterns of each user is different. Because each user has a unique rating pattern, a rating score needs to be interpreted. Even if two users give the same item the same score, it does not always indicate they like it to the same extent if their rating patterns are different. As a result, utilizing the actual ratings from users who rate items within different ranges to predict the rating score of the target user is ineffective and may result in low recommendation accuracy. Because of this

issue, some researchers have adjusted the ratings from several ranges to match the common range in advance [4, 15, 43, 44, 50]. These methods can be called *rating conversion*.

Additionally, when applying the neighborhood-based idea into model-based CF, neighbors (i.e., friends and historical items) are utilized in the model, so we should carefully consider the effects of neighbors to the model. In the real world, people tend to believe their friends' opinions when deciding to choose something. Naturally, people do not believe all friends' opinions with the same level, closed friends or best friends always have more influences than regular friends. This point is one point to consider as well.

This chapter consists of the details of three main proposed methods to handle the improper rating-range problem which are 1) *Integrating the importance levels of friends into trust-based ant-colony recommender systems* (TrustAnt) in [72], 2) *Translation-based Embedding Model for Rating Conversion in RSs* (TransRS) in [73], and 3) *Attentive Hybrid Collaborative Filtering for rating conversion in RSs* (AHCF) in [74].

5.2 Trust-based Ant Recommender Systems

TrustAnt [72] is a trust-based RSs which is the extended version of *Applying Ant-Colony Concepts to Trust-Based RSs* [71]. This model is categorized in the neighborhood-based CF RS and it consists of neighbors selection method and rating conversion methods. The details of this model are as follows.

5.2.1 Trust Network and Input Representation

In this subsection, the details of trust network and input representation, which are used in the first step of this system, are described as follows.

Create Trust Network

The trust network is modeled as a directed graph $G = (V, E)$, where the set V of nodes represents users and the set E of directed edges represent trust statements issued among the users. The input data comprises two matrices.

- $[N \times M]$ item-ratings matrix, a matrix that comprises ratings that users have given to various items in the past, with N rows for the users in the system and M columns for the items in the system.

- $[N \times N]$ user-trust matrix, a matrix that holds the trust statements issued between pairs of users in the system, with the rows being the source users issuing the trust statements and the columns representing the target users about whom trust statements are issued.

Compute Trust Value

When the trust network is created, a trust value is assigned to each edge. In much of the research, the trust value is computed in terms of similarity and confidence, usually involving co–rated items. Problems will occur if the data is sparse, and using the similarity value as a trust value will lead to the problem of 'trust symmetry'.

If the similarity between user A and B is equal to 0.5, current systems will assign a trust value of 0.5 from both user A to user B and user B to user A. In real life, even if user A trusts user B, user B might not trust user A. To solve these problems, I apply the solutions proposed in [15], to compute the trust value. To reduce the data-sparsity problem, the trust value is calculated based on features of users and items using the single-value-decomposition (SVD) method. This decomposes a user-rating matrix R into three matrices:

$$R = USV^t \tag{5.2.1}$$

where U is the user matrix, S is the reduced matrix, and V^t is the transpose matrix of the item matrix. Each row of matrix U represents a user feature vector. It contains the latent features presenting the characteristics of each user. Cosine similarity is then applied to these feature vectors to find the similarity of every pair of users:

$$sim_{A,B} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{5.2.2}$$

where A_i is the i^{th} latent feature of user A (the target user), B_i is the i^{th} latent feature of user B, and n is the number of latent features.

To overcome the trust-symmetry problem, [15] proposed to exploit both the similarity between a pair of users and the reliability of user B. The "trustworthiness" of user B is calculated as follows:

$$trustworthiness_B = \frac{ln(n_{in,B} + e)}{ln(max(\{n_{in,C} | C \in \{userintrustnetwork\}\}) + e}, \tag{5.2.3}$$

where e is a natural number, $n_{in,B}$ is the number of in-degree edges for user B, and $max(\{n_{in,C} | C \in \{userintrustnetwork\}\})$ is the maximum number of in-degree edges among users in the trust network.

In this proposed model, I propose a new equation to calculate the trust value from A towards B by merging similarity and trustworthiness from the previous calculation using the harmonic mean:

$$trust_{A\to B} = \frac{2 \times trustworthiness_B \times sim_{A,B}}{trustworthiness_B + sim_{A,B}}. \tag{5.2.4}$$

5.2.2 Ant algorithm

After obtaining the trust network and input representation, the next step is finding the set of raters for a target user. In this step, I apply the ACO approach. According to the properties of trust, a trust value is generated by a human, so it can change over time.

To deal with the dynamic nature of the trust value, I apply the ACO approach to this step using '*pheromone-update*' strategy that can update trust values. I will call this the "ant algorithm."

Ant's Edge Selection

The purpose of this step in the process is to find the set of raters who have high pheromone levels. The raters being considered are users who are in the target user's network (web of trust) and have a rating r for the target item i. This process is inspired by ALT-BAR [10]. For example, assume that user u_1 is a rater for the target user A and always rates items with scores in the range 1–3, whereas user A always scores in the range 3–5. That is, target user A's rating pattern and user u_1's rating pattern involve different ranges. Before calculating predictions, the model should convert the rater's rating to match the target user's perspective.

To understand this more precisely, I will describe the first step of this process. The system first generates k ants, locating them at the node of the target user, and then calculates the probability of crossing edges connecting to users in the target user's web of trust. Then, the system chooses to cross edges connecting to the user y having the highest probability and the ant will move to the selected node. Probabilities are calculated by this following equation:

$$p_{xy}^k = \frac{(\tau_{xy})^\alpha (\eta_{xy})^\beta}{\sum_{z \in N_x^k} (\tau_{xz})^\alpha (\eta_{xz})^\beta}, \tag{5.2.5}$$

where τ_{xy} is the pheromone level on the edge xy, η_{xy} is the trust value that user x express towards user y, and α and β are parameters that control the influence of τ_{xy} and η_{xy}, respectively.

As for ALT-BAR [10], this proposed method extends the path (solution) for each of the artificial ants at each iteration. The process stops when all ant paths have either reached a length d or have reached a user outside the web of trust (i.e., no more edges can be traversed).

This proposed method is similar to ALT-BAR, in that each artificial ant stops constructing its path (solution) within an iteration either when each ant path is of length d or when the ant reaches a user outside the web of trust. When all k ants stop constructing their paths, it means that the system has completed one iteration. The process then reiterates by dispatching the k ants again from the target user node, stopping when the system reaches a certain number of iterations t. The last iteration of the process will involve k solutions containing raters. Note that not every solution will contain raters, because raters are the subset of users who have ratings r for the target item i.

Pheromone-Update Mechanism

In this step, I apply ACS in the same way as does ALT-BAR [10]. The pheromone-update model of ACS is separated into two parts, namely the local pheromone update and the global pheromone update.

Local pheromone update: the pheromone-update mechanism that occurs when each ant k traverses an edge xy. As for ALT-BAR, the pheromone level is adjusted by:

$$\tau_{xy} = (1-\rho)(\tau_{xy}) + (\rho)(\tau^0_{xy}), \tag{5.2.6}$$

where ρ is the pheromone evaporation coefficient, usually set to 0.1, and τ^0_{xy} is the initial pheromone level on xy, computed by:

$$\tau^0_{xy} = \frac{T_{xy}}{n(u_x)\sum_{z \in WOT_x} T_{xz}}, \tag{5.2.7}$$

where T_xy is the trust value that user x expresses towards user y, $n(u_x)$ is the number of users in the web of trust for user x, and WOT_x is the web of trust for user x.

Global pheromone update: in an ACS, this pheromone–update mechanism takes place after all ants k finish constructing their paths. That is, global pheromone update occurs at the end of each iteration. The global pheromone–update strategy differs from the local pheromone–update strategy in that not all edges will be updated, but only the edges that belong to the best paths constructed in that iteration.

In ACS, the global pheromone update is usually given by:

$$\tau_{xy} = (1-\rho)(\tau_x y) + \Delta Q, \tag{5.2.8}$$

where ΔQ is a small quantity that will increase the pheromone level on each edge. In general, ΔQ's goal is to increase pheromone levels on edges. The pheromone level on the edge will be high when the paths that are to be updated involve a high value for ΔQ.

In ALT-BAR, the co-rated item is used as one of the factors in updating the global pheromone value. However, this can lead to the data-sparsity problem. This proposed model aims to solve this problem by proposing a new global pheromone-update equation.

The proposed equation is inspired by Bedi et al. [7] and Kaleroun et al. [47]. They use three main factors in this process to calculate the ΔQ value, namely the pheromone level on the edge, the distances between the active user and other users, and the number of items rated by each user that are unrated by the active user. These three factors are important but they are not sufficient to calculate and generate the most accurate solution. To improve the accuracy, I propose using an additional factor, namely the popularity of each user. If a user is very popular (many users in the user's trust network), the directed edge to that user should have a high pheromone level because that user is a reliable user. I therefore use the original three factors from [7] and the popularity of the user to calculate ΔQ as:

$$\Delta Q = \frac{2 \cdot (\tau_{xy} \cdot \frac{1}{d_{Ay}} \cdot \frac{T_{traced_y}}{T_{unrated_A}}) \cdot (\frac{\sum_{y \in WOT_z} T_{zy}}{n(z)})}{(\tau_{xy} \cdot \frac{1}{d_{Ay}} \cdot \frac{T_{traced_y}}{T_{unrated_A}}) + (\frac{\sum_{y \in WOT_z} T_{zy}}{n(z)})}, \tag{5.2.9}$$

where τ_{xy} is the pheromone level on the edge between user x and user y, d_{Ay} is the distance between the target user A and user y, T_{traced_y} is the number of items traced by the user at node y that are unrated by node A, and $T_{unrated_A}$ is the total number of items unrated by node A.

As noted above, current global pheromone–update methods do not involve every constructed path but only the best path(s) in each iteration, with the update formula varying from method to method. These methods select paths with high pheromone levels or that contain reliable users. However, reliable users may not have ratings for the target item. That means that the best paths may not contain raters. If the system cannot obtain a rater in this step, the system cannot generate recommendations.

I propose using an optimization condition in the global pheromone–update process that limits updating to only paths that contain raters. This proposed condition will increase the likelihood that paths that contain raters will have higher pheromone levels than paths without raters. It will also lead to greater coverage because it ensures that the system will obtain more raters from this step than the optimization conditions in current ant-based recommender methods.

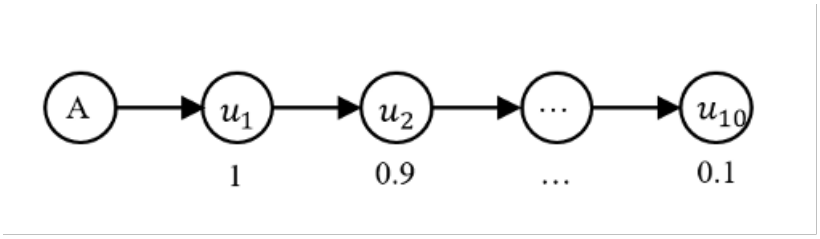

Figure 5.1: The constant assigned weight of each raters.

5.2.3 Rating Conversion and Weight Adjustment

Rating Conversion

In existing systems [7, 10, 47], the predicted rating is usually calculated by averaging the actual ratings given by raters. using a rater's actual rating causes a problem when the rater's rating pattern and the target user's rating pattern involve different ranges.

To convert a rating to the target user's perspective by using a relation between each user and the target user, I apply a method based on that of Chalermpornpong et al. [15], who proposed the rating transpose function, which is called 'W's transpose function' as shown in Eq. 3.3.3. This is adjusted by including a confidence term and a distribution term.

Weight Adjustment

In existing systems, the predicted rating for the target user is usually calculated without considering the impact of each rater on the target user. In the real world, a user who is directly trusted by the target user will have more influence on the target user than a user who is further away from the target user. Therefore, raters should have different levels of impact on the target user.

In the first version of this model presented in [71], I assigned a weight to each rater before generating the recommendation. These weights decrease as raters are further away from the target user. The final rating score $W'_{u\to A}(r_{ui})$ from the rater is calculated by multiplying the weight γ by the transposed rating $W_{u\to A}(r_{ui})$ as follows.

$$W'_{u\to A}(r_{ui}) = \gamma \times W_{u\to A}(r_{ui}) \tag{5.2.10}$$

In the previous version [71], it used the constant weights. For example, if the depth d is 10, the distance coefficient was calculated as follows. If the distance from the active user is one, γ is one. If the distance from the active user is two, γ is 0.9 and so on, until the distance from the active user is 10, when γ is 0.1, as shown in Figure 5.1. Likewise, with the depth d of search set as is 20 in the experiments, γ will be $\{1, 0.95, 0.9, \ldots, 0.1, 0.05\}$.

Apart from setting the weight γ as a constant value, in [72], I propose another way to calculate the weight. Constant weights are not suitable for the system if the depth d of the search is changed. Therefore, I propose a new weight adjustment inspired by the concept "all friends are not equal". The concept of "weights in social graphs to improve search" was proposed by Hangal et al. [36]. They proposed that searches in social networks can be made more effective by incorporating weighted and directed influence edges in the social graph. Because this model derives trust-based RSs, the directed influence edges in this model should be calculated by the trust value.

Instead of using a constant weight for each rater, I propose to use trust value as a factor in calculating the weight, called a trust weight. The weight that will be assigned to each rater will be calculated from a trust value and a discount factor based on the distance between that rater and the active user. The equation to calculate weight for each rater is inspired by [36]. This model calculates the weight of rater u as follows. For a path P that contains edges $e_1, e_2, ..., e_n$, where n is the level of rater u:

$$\gamma_u = \prod_{i=1}^{n} (trust_{e_i} \cdot D) \tag{5.2.11}$$

where $e_i \in P$ and D is a discount factor, which I set to 0.9.

From this equation, because the decay factor decreases with each hop, the model will obtain a new weight for each rater. Therefore, a rater who is further away from the active user will receive a lower weight than one who is closer to the active user. Note that the transposed rating of each rater will be adjusted by the weight obtained from Eq. 5.2.11.

5.3 Translation-based Embedding Model for Rating Conversion in Recommender Systems

This subsection demonstrates how to select neighbors and apply a rating conversion method that proposed in TransRS [73].

5.3.1 Neighbor Selections

As most of the RSs use the information or opinion from the target user's friends or raters to generate the recommendations, in TransRS [73], I used n best rater(s) of the target user in the prediction step. Hence, the model first finds the n best raters of a target user.

Inspired by CPE assumption [77],

if u_1 and u_2 have a similar preference for the same item i, then the vectors of u_1 and u_2 should be close.

For an item $i \in I$, let U_i denote the set of users who rated i. Then, for a user u and an item $i \in I$, his/her best rater regarding i is defined as

$$bf(u,i) \equiv \text{argmin}_{u' \in U_i} \|\vec{v}^*_u - \vec{v}^*_{u'}\|_2. \tag{5.3.1}$$

Based on the assumption above, this equation indicates that the best rater is the user who has the minimum distance from the target user in the vector space.

After obtaining the set of best raters, the rating conversion will be performed on users in this set. The details of rating conversion is described in the next subsection.

5.3.2 PCA-based Rating Score Conversion

This subsection proposes how to convert rating scores between a pair of users from the obtained best raters from the previous subsection, which is also proposed in TransRS [73]. There are three main steps in order to achieve this: 1) obtaining user-rating matrix, 2) rating pattern extraction, and 3) rating score conversion.

User Rating Matrix

For a triple of user u, item i, and rating score s: (u,i,s) in the training data, let us consider the following vector:

$$\vec{r}_{u,i} \equiv \vec{v}^*_i - \vec{v}^*_u \,. \tag{5.3.2}$$

This vector is a rating score vector derived from the user and the item embedding vectors. It is expected to be close to $\vec{r}^*_s$ and referred to an *approximate rating vector*.

This model obtains a user's profile; in other words, a user's preference. The user's profile represents how user rates items. User's profile can be constructed from the approximate rating vectors. For a user u, let us consider the following matrix consisted of u's approximate rating vectors:

$$\vec{R}_u \equiv (\vec{r}_{u,i})^T_{i \in I_u} \tag{5.3.3}$$

where I_u denotes the set of items rated by u and T denotes the transpose of a matrix. Each row of $\vec{R}_u$ is a row vector of an approximate rating vector of the user u and an item i in I_u, i.e. $\vec{r}^T_{u,i}$. I refer to $\vec{R}_u$ as *a user rating matrix*.

5.3.3 Rating Pattern Extraction

After that, I apply the Principal Component Analysis (PCA) [67] to the user rating matrix to extract each user's rating pattern. There are two benefits of PCA in RSs: 1) reducing the dimensionality and 2) finding patterns in the high-dimensional dataset in order to extract the most crucial characteristics from the data. In this method, I take the latter advantage of PCA to extract the rating patterns of each user from the user rating matrix.

Following PCA principle [67], I first create the normalized user rating matrix. For a user u, the mean vector of u's approximate rating vectors is given by

$$\vec{\mu}_u = \frac{1}{|I_u|} \sum_{i \in I_u} \vec{r}_{u,i} \ . \tag{5.3.4}$$

Using the mean vector, the normalized user rating matrix is given by

$$\bar{\vec{R}}_u \equiv (\vec{r}_{u,i} - \vec{\mu}_u)^T_{i \in I_u} \ . \tag{5.3.5}$$

Then, I create *a feature matrix* $\vec{Q}_u$ which represents u's rating pattern and calculate the eigenvectors of the covariance matrix of $\bar{\vec{R}}_u$. After computing all eigenvectors, sort the eigenvectors in descending order of their eigenvalues. The first row of $\vec{Q}_u$ has the highest eigenvalue.

5.3.4 Score Conversion

For users a and b, let us convert user b's rating on target item i into user a's perspective. First, the approximate rating vector $\vec{r}_{b,i}$ of user b on item i is normalized by b's mean vector. Then it is converted to the user a's perspective. The model makes projection onto user a's space by multiplying with user a's feature matrix $\vec{Q}_a$. Finally, the derived rating score is moved to the original scale by a's mean vector $\vec{\mu}_a$. The converted rating score is given by

$$\vec{r}_{b,i,a} = \vec{Q}_a(\vec{r}_{b,i} - \vec{\mu}_b)^T + \vec{\mu}_a. \tag{5.3.6}$$

To this point, the converted rating score of target user's n best rates is obtained and will be used for the prediction step which will be presented in Chapter 7.3.

5.4 Attentive Hybrid Collaborative Filtering

In the previous subsection, I have described the details of neighbors selection and rating conversion using a translation-based embedding model proposed in TransRS [73]. In this subsection, the details of neighbor selection and rating conversion based on attention mechanism proposed in AHCF [74] will be described as follows.

5.4.1 Friends and Historical Items Selection

Before going into the details of rating conversion in the friend module, I would like to firstly introduce the friend and historical item selection method. Since the numbers of friends and historical items of each target user are different, and treating all friends and all historical items at the same level of influence on the target user–item pair is not appropriate because it might lead to low accuracy prediction. Thus, the model needs to select the high-quality friends and high-quality historical items to participate more role in the model than the low-quality users and items. In neighborhood-based CF RSs, similarity levels are widely used for the friend and item selection methods. According to real-world assumption that close friends should have more influence on the target user than regular friends, so the user (resp. item) who is similar or close to the target user (resp. target item) should have more power than the user (resp. the item) who is not similar to the target user (resp. the target item). In order to solve this issue, this model adopts an *attention mechanism* [75] that has been widely used in many fields and is successful for assigning nonuniform weights [16–18]. This can help model to adaptively learn the influence of each friend and historical item on the target user–item pair.

Inspired by SAMN [17], which adopts the attention mechanism into their model to adaptively learn the influence strength of each friend. The goal of their friend-level attention is to assign non-uniform weights to users' friends, and the weights are varied when the user interacts with different items. Therefore, I adopt the same kind of attention layer as in SAMN [17] to assign non-uniform weights (i.e., similarity score) to users' friends and historical items.

This friend and historical item selection modules are shown in Figure 5.2. The full architecture is shown in Figure 7.3 in Chapter 7.4.4.

Friend selection

According to its protocol, a two-layer network is applied to compute the attention score. For the friend side, the attention score α_f can be computed by using the UR-matrix of the target user u and the UR-matrix of each friend f. Consider the set F_v of friends with respect to the

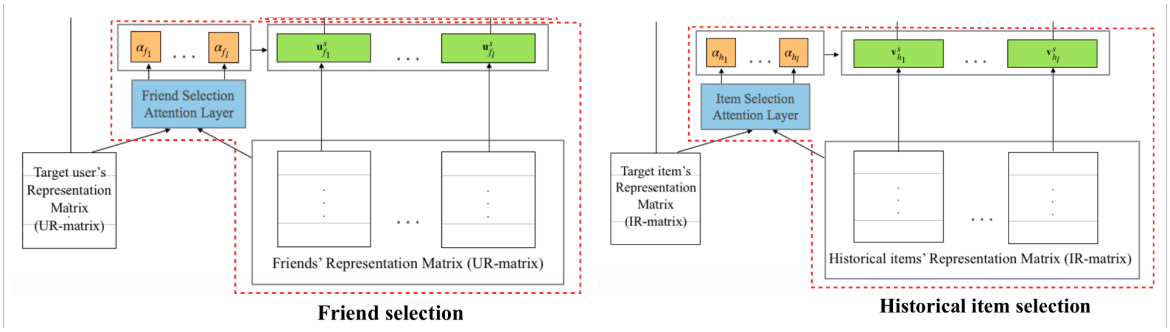

Figure 5.2: Friend selection and historical item selection module

target item v. Let $f \in F_v$ be a friend in a user–item pair $\langle u, v \rangle$. The scoring function of the friend selection part is then defined as

$$\beta_f = \mathbf{h}_u^T ReLU(\mathbf{Wu}_1 \mathbf{U}_u + \mathbf{Wu}_2 \mathbf{U}_f), \tag{5.4.1}$$

where $\mathbf{Wu}_1, \mathbf{Wu}_2 \in \mathbb{R}^{d \times a_f}$ are the first layer model parameters, $\mathbf{h_u} \in \mathbb{R}^{a_f}$, is the second layer model parameter, a_f denotes the dimensionality of the Friend Selection Attention Layer, and *ReLU* is a nonlinear activation function.

The weight for each friend is then obtained by normalizing the attention weights using the softmax function, as is the common practice for neural attention networks. The attention score can then be calculated as

$$\alpha_f = \frac{exp(\beta_f)}{\sum_{j \in F_v} exp(\beta_j)}. \tag{5.4.2}$$

Suppose this friend has rated the target item v with a score s and $\mathbf{u}_f^s \in \mathbf{U}_f$ is the s-th row of friend f's UR-matrix. After obtaining the friend attention weight, then the model will multiply the obtained weight α_f with $\mathbf{u}_f^s$ before feeding it into the friend module. The weighted friend $\overline{\mathbf{u}}_f^s$ can be calculated as

$$\overline{\mathbf{u}}_f^s = \alpha_f \mathbf{u}_f^s. \tag{5.4.3}$$

For more understanding, I will give some concrete examples in accord with the illustration in Figure 7.4. Suppose that there are 2 users who have rated the target item which are user A and user B. They rated the target item with 5 and 3 score respectively. These two users and the target user's UR-matrices will be fed into the friend selection module. After calculating the weight of each user by Eq. 5.4.2, suppose that user A's weight is 0.7 while user B's weight is 0.3. It means user A is more similar to the target user, so it leads to more influence on the target user than user B, so user A will have more weight in calculation. After the

model gets the weight of each user, I multiply the weight with their vector of representation matrix. In the case of A, it will be the 5th row of her matrix, while it will be the 3rd row of user B's matrix. Then these weighted friends' vectors will be fed into the friend module.

Historical Item Selection

Likewise, for the historical item side, the attention score α_h can be computed by using the IR-matrix of the target item v and the IR-matrix of each historical item h. Consider the set of historical items with respect to the target user u (the items that have been rated by the target user u in the past), denoted as H_u. Let $h \in H_u$ be a historical item in a user–item pair $\langle u, v \rangle$. The scoring function of the item selection part is then defined as

$$\beta_h = \mathbf{h}_i^T ReLU(\mathbf{Wi}_1 \mathbf{V}_v + \mathbf{Wi}_2 \mathbf{V}_h), \tag{5.4.4}$$

where $\mathbf{Wi}_1, \mathbf{Wi}_2 \in \mathbb{R}^{d \times a_f}$ are the first layer model parameters, and $\mathbf{h_i} \in \mathbb{R}^{a_f}$ is the second layer model parameters.

The weight for each item is then obtained by normalizing the attention weights using the Softmax function. The attention score can then be calculated as

$$\alpha_h = \frac{exp(\beta_h)}{\sum_{j \in H_u} exp(\beta_j)}, \tag{5.4.5}$$

where H_u is the set of items that target user u has been rated in the past.

Suppose this item has been rated by the target user u with a score s and $\mathbf{v}_h^s \in \mathbf{V}_h$ is the s-th row of item h's IR-matrix. After obtaining the item attention weight, then the model will multiply the obtained weight α_h with $\mathbf{v}_h^s$ before feeding it into the item module. The weighted historical item $\overline{\mathbf{v}}_h^s$ can be calculated as

$$\overline{\mathbf{v}}_h^s = \alpha_h \mathbf{v}_h^s. \tag{5.4.6}$$

According to the illustration in Figure 7.4, suppose that the target user has rated 2 movies in the past, which are movie X and movie Y. She rated these movies with 2 and 4 score respectively. These 2 historical items and the target item's IR-matrices will be fed into the historical item selection module. After calculating the weight of each item by Eq. 5.4.5, suppose that movie X's weight is 0.2 while movie Y's weight is 0.8. It means item Y is more similar to the target item than item X. Based on the assumption that the target user shall prefer an item that is similar to their historical item, in this case, the target user shall prefer the target item because she rated item Y with 4 score and it is similar to the target item, so she tends to give around a 4-score to the target item. After obtaining the weight of each item,

I then multiply the weight with their vector of representation matrix. In the case of item X, it will be the 2nd row of its IR-matrix, while it will be the 4th row of item Y's IR-matrix. Then these weighted historical items' vectors will be fed into the item module.

5.4.2 The Friend Module

This subsection describes the details of how to do rating conversion in end-to-end hybrid CF. I have proposed a module named **the Friend Module**, which aims to convert the target user's friend vector to match the target user's perspective and assign different weights for individual users. This Friend Module is implemented in AHCF [74].

The input of this module is the friends' embeddings that can be obtained from the previous friend selection module (Eq. 5.4.3). The output of the Friend Module is a summation of the weighted projected vectors of the target user's friends. To convert friend vectors, I introduce a user-perspective unit matrix (UPU-matrix) to the model.

Because users have their own individual perspectives on items, the way they rate items are all different, so it can be inferred that they have their own rating patterns. In most of traditional neighborhood-based CF, this issue is ignored, so they did not consider the different rating patterns of each user. Since my assumption is that rating patterns of each user are different and to use a friend's rating to predict the rating of the target user towards the target item, friend's rating needs to be converted into the target's user perspective first.

In [61], a *local context unit* is learned in addition to word embedding to utilize the word-specific influences of each word on its context words. Inspired by this [61], this proposed module aims to convert friend vectors to match the target user's perspective. In other words, this module needs to learn the specific influences of each friend on the target user. I then adopt this idea from [61] and introduce a UPU-matrix in addition to the UR-matrix and IR-matrix. I design the UPU-matrix to represent each user's individual perspective. Specifically, each column of the matrix captures a rich range of user preferences. It captures not only the user's overall preference but also preferences for specific rating scores. This matrix is learned to utilize the score-specific influences of each friend on the target user. It then enables the model to convert friend vectors to best match the target user's perspective. The UPU-matrix for user u is denoted by $\mathbf{K}_u \in \mathbb{R}^{d \times z}$.

After the friend selection step, the model will obtain a set of $\overline{\mathbf{u}}_f^s$, which is a set of weighted friends' representation vectors on s score. If $\overline{\mathbf{u}}_f^s \in U_f$ is the s-th row of friend f's UR-matrix and $\mathbf{k}_u^s \in \mathbf{K}_u$ is the s-th column of target user u's UPU-matrix, the projected vector of friend f with respect to the target user u is calculated by

$$\mathbf{p}_f = \mathbf{k}_u^s \odot \overline{\mathbf{u}}_f^s, \tag{5.4.7}$$

where $\odot$ is the Hadamard product (i.e., element-wise multiplication).

This equation converts each friend's representation vector on a score s so as to match the target user's perspective. In this step, the model calculates the projected vector $\mathbf{p}_f$ for each friend in F_v. The set of friends' projected vectors with respect to the target user's perspective is then input into the Friend Module Attention Layer for weight assignment.

The Friend Module Attention Layer

Because the importance of all friends on a pair of each target user–item is not the same, they should have different influence levels with respect to the target user, and so any weighting (to represent relative importance) should also vary. The function of this Friend Module Attention Layer is different from the previous attention layer. In this step, I would like to vary the weight for users interacting with different items. The same friend might have a different influence on the target user when the target item is different. According to the attention mechanism's protocol, a two-layer network is applied to compute the attention score α_f, using the target user u, target item v, and projected friend vector $\mathbf{p}_f$ as inputs. The scoring function is then defined as

$$\beta_{p_f} = \mathbf{h}^T ReLU(\mathbf{W}_1\mathbf{u} + \mathbf{W}_2\mathbf{p}_f + \mathbf{W}_3\mathbf{v}), \tag{5.4.8}$$

where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3 \in \mathbb{R}^{d\times a}$ are the first layer model parameters, $\mathbf{h} \in \mathbb{R}^a$ is the second layer model parameter, a denotes the dimensionality of the Friend Module Attention Layer.

The final weight for each friend is then obtained by normalizing the attention weights using the softmax function. The attention score can then be calculated as

$$\alpha_{p_f} = \frac{exp(\beta_{p_f})}{\sum_{j\in F_v} exp(\beta_{p_j})}, \tag{5.4.9}$$

where F_v is all friends of user u on the view of item v.

After obtaining the attention weight for each friend, the output of the Attention Layer represents the summation of the projected vector of friends $\mathbf{p}_{F_{(u,v)}}$. This summation vector represents all friends' vectors from the target user's perspective, with each friend having an individual weight representing importance to the target user. It can be calculated as

$$\mathbf{p}_{F_{(u,v)}} = \sum_{f\in F_v} \alpha_{p_f}\mathbf{p}_f. \tag{5.4.10}$$

The illustration of the friend module is shown in Figure 5.3 which is a part of full architecture Figure 7.3 in Chapter 7.4.

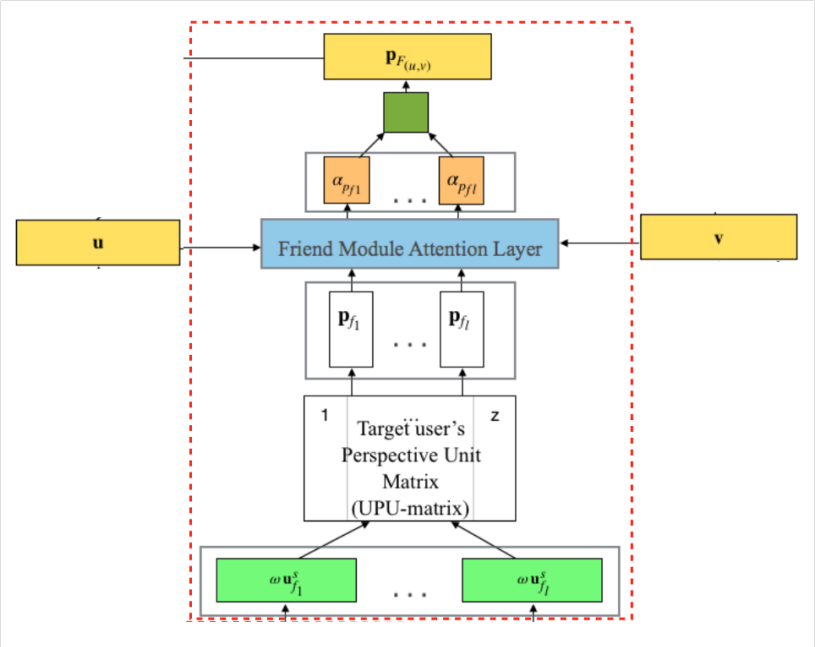

Figure 5.3: The illustration of the friend module.

For better understanding, I would like to continue with a concrete example according to the illustration in Figure 7.4. After the model obtains the weighted friend's vector from the friend selection module, it will be fed into the Friend module. From this example, user A rated the target item with 5 score, so her weighted vector will be multiplied by the 5-th column of the target user's UPU-matrix for rating conversion (Eq. 5.4.7). The model will do this to all friends, so user B rated target item with 3 score, so his weighted vector will be multiplied by the 3rd column of the target user's UPU-matrix (Eq. 5.4.7). To this point, the model has projected the vector of each friend from the target user's perspective. After that, these projected vectors of each friend will go through the friend module attention for weight assignment (Eq. 5.4.8 and 5.4.9). Suppose that the attention weight that is calculated from Eq. 5.4.9 of user A and B are 0.8 and 0.2, respectively. Then the model will calculate the summation vector that represents all friends' vectors from the target user's perspective by Eq. 5.4.10, with each friend having an individual weight representing importance to the target user. In this case, it means that user A will have more weight and influence than user B from the target user's perspective.

5.4.3 The Item Module

Apart from user-based CF, in this model, I also consider item-based CF. Some users may have few friends, so the model may lack information about those users and lead to low accuracy

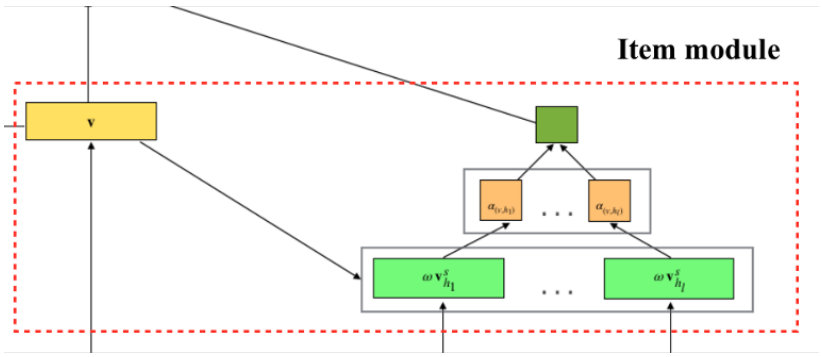

Figure 5.4: The illustration of the item module.

prediction. Therefore, I also introduce item-based CF to this model to handle that kind of user, effectively. Moreover, the friend module only focuses on 'how users rate items'. In addition to the friend module, I think we should also consider 'how items are rated by users' too. If the model can specify more detail on both the user and item side, the accuracy of the model might increase.

The inputs of this module are target item v and a set of weighted historical items of target user which can be obtained from Eq. 5.4.6.

First, aggregate the target item v's IR-matrix $\mathbf{V}_v$ by calculating the means of each column of the target item's IR-matrix, represented by $\mathbf{v}_v \in \mathbb{R}^d$. Then, the similarities between the target item v and weighted historical items of target user h can be computed by their dot products as

$$sim_{(v,h)} = \mathbf{v}_v^T \overline{\mathbf{v}}_h^s, \tag{5.4.11}$$

where $\overline{\mathbf{v}}_h^s$ is a weighted historical item that has been rated with a score s by the target user in the past. This score computes the compatibility between the target item and the historical items consumed by the user.

Then, the model will normalize similarity scores to be a probability distribution over historical items by softmax function: $\alpha_{(v,h)} = \text{Softmax}(sim_{(v,h)})$. Finally, the output of this item module can be calculated by

$$\mathbf{t}_{H_u} = RELU\Big(\sum_{h \in H_u} \alpha_{(v,h)} \overline{\mathbf{v}}_h^s\Big). \tag{5.4.12}$$

The illustration of the item module is shown in Figure 5.4 which is a part of full architecture Figure 7.3 in Chapter 7.4.

For better understanding, I continue to explain the concrete examples. After the model obtains the weighted historical item's vector from the historical item selection module, it

will be fed into the Item module. From this example, the target item's IR-matrix will be aggregated by calculating means of each column. Then, the similarities between the target item and item X will be calculated by Eq. 5.4.11. The model will do this similarity calculation between the target item and each historical item. Then, the model will normalize similarity scores to be a probability distribution over historical items by softmax function. From this example, suppose that the similarity score of the target item and item X is 0.3 while the similarity score of the target item and item Y is 0.7. Finally, the output of this item module can be calculated by Eq. 5.4.12 which indicates that historical item Y plays a more important role in this calculation and the target user tends to rate the target item with a similar score that she gave to item Y.

5.4.4 Generating Latent Relations

After the model obtains the summation of friends from the friend module (Eq. 5.4.10), the model then combines it with the aggregated target user representation vector $\mathbf{u} \in \mathbb{R}^d$. This vector is calculated as the means of each column of the UR-matrix. The combination of the aggregated target-user representation and the summation of friends' weighted projected vectors can be calculated as

$$\mathbf{u}^* = \lambda \mathbf{u} + (1-\lambda)\mathbf{p}_{F_{(u,v)}}, \tag{5.4.13}$$

where λ is a parameter which controls the weight between user-based and model-based of the user part.

Likewise, after obtaining the summation of historical items from the item module (Eq. 5.4.12), the model then combines it with the aggregated target item representation vector $\mathbf{v} \in \mathbb{R}^d$. This vector is calculated as the means of each column of the IR-matrix. The combination of the aggregated target item representation and the summation of historical items' weighted vectors can be calculated as

$$\mathbf{v}^* = \theta \mathbf{v} + (1-\theta)\mathbf{t}_{H_u}, \tag{5.4.14}$$

where θ is a parameter which controls the weight between neighborhood-based and model-based of the item part.

Finally, the model generates a latent relation vector $\mathbf{r}$ for each user–item pair by multiplying combined user $\mathbf{u}^*$ and combined item $\mathbf{v}^*$ as

$$\mathbf{r} = \mathbf{u}^* \odot \mathbf{v}^*. \tag{5.4.15}$$

Table 5.1: Comparison of the model features between TrustAnt [72], TransRS [73], AHCF [74], and other rating conversion methods

Methods	TrustAnt	TransRS	AHCF	Other
Ability	convert rating in scalar form	convert rating in vector form	convert rating in vector form	convert rating in scalar form
Input	scalar rating	pre-trained embeddings from previous module	optimized embedding from previous iteration	scalar rating
Output	scalar form	vector form	vector form	scalar form
Optimization	ACO optimization	separately	end-to-end	no optimization

This latent relation vector $\mathbf{r}$ is dependent on user and item and is learned to best explain the relationship between a pair of user–item. Note that the latent relation vector $\mathbf{r}$ here is different from r in LRML, which generates their r from only the target user–item pair. My proposed latent relation vector $\mathbf{r}$ is generated from $\mathbf{u}^*$ and $\mathbf{v}^*$ which are respectively derived from the combination between the target user and friends, and the target item and target user's historical items.

5.5 Discussion

According to the previous subsections which describe details of neighbors selection methods and rating conversion methods of TrustAnt [72], TransRS [73] and AHCF [74], in this subsection, I will discuss on the similarities and differences among them.

5.5.1 Comparison between Rating Conversion Methods

All rating conversion methods in RSs share the same objective which is converting one's rating score into another one's perspective, then utilize the converted score in the prediction step. However, each of them has its own characteristics in various aspects. The brief comparison is described in Table 5.1.

Ability to Convert Rating Score

Typically, rating conversion methods are proposed to solve the improper rating-range problem which occurs in the the neighborhood-based CF model. The input of the neighborhood-based

CF model is usually in scalar form and it does not learn any parameter using optimization algorithm, so it does not contain any embedding. Thus, most of the related rating conversion methods can only convert rating score in the form of scalar. They cannot convert rating score in vector form including the proposed rating conversion method in TrustAnt [72], which is the neighborhood-based CF model. TrustAnt's rating conversion method can only convert rating score in scalar form.

In contrast, this thesis focuses on hybrid CF in which the rating score in the proposed model is not in a form of scalar like in the traditional neighborhood-based CF model. Both rating conversion methods of TransRS [73] and AHCF [74] convert the rating score of one user into another user's perspective in vector form. To the best of my knowledge, there is no prior research studying the rating conversion in vector form.

However, even though these two proposed rating conversion methods can convert rating score in the vector form, they perform their tasks differently. Given a pair of target user A and his/her friend B, the rating conversion method in TranRS [73] can convert rating score vector from friend B into the target user A's perspective if we know a approximate rating vector of user B towards the target item. These approximate rating vectors of users towards items have to be learned from the previous TransE module, which is independent from the rating conversion step.

On the other hand, in case of AHCF [74] which is an end-to-end NN CF model, the rating conversion is performed in a different way from in TransRS [73]. In this model, the rating conversion step is implemented and embedded within the end-to-end model, meaning that it does not require pre-trained embeddings like in TransRS [73]. This rating conversion can be done within the model itself in one single optimization step.

Input of Rating Conversion Methods

In the aspect of input, the existing rating conversion methods always convert rating score in scalar form, so their input will be a scalar rating score. As well as in TrustAnt [72], whose input is in scalar form.

In contrast, since my proposed rating conversion methods convert rating score in a vector form, so the input of my proposed method are embedding vectors, not a scalar. As compared to the scalar input, the vector input contains more meaningful information which is consequently more effective in the optimization and rating prediction procedures.

Even though input of both of my proposed rating conversion methods is in vector form, input used in TransRS [73] and AHCF [74] are different. Because in TransRS [73], I use the existing TransE embedding model to train users, items, and ratings embeddings as the first step of the model, then feed the obtained output into the rating conversion step. Therefore,

the input of the rating conversion step is the embeddings obtained from TransE embedding model, which can be catagorized as pre-trained embeddings.

While in AHCF [74], the input of the rating conversion step is different from TransRS [73]. Because AHCF is an end-to-end NN CF model, it optimizes itself through one single step. In other words, the input of the rating conversion step is embeddings that are optimized through rating prediction.

Output of Rating Conversion Methods

Apart from the input of each rating conversion, the output of them are also different as well. In existing rating conversion methods and TrustAnt [72], the output is always in scalar form. Then, this obtained scalar output will be used in the prediction step. On the contrary, in my proposed rating conversion methods TransRS [73] and AHCF [74], the output obtained from the rating conversion step is in vector form and then will be used in the next step of each model.

Optimization

Now let me discuss the optimization of model parameters. Generally, the existing rating conversion methods do not require optimization since they do not learn any parameters. However, in TrustAnt [72], it adopts the ACO concept, so the optimization in TrustAnt will be the same as the ACO.

As TransRS [73] is a building blocks model which consists of three components, they separately do their own task in their own building block (i.e, module). Therefore, in TransRS [73], the model-based module optimizes within itself, and the neighborhood-based module has no optimization to perform. On the other hand, AHCF [74] is an end-to-end NN CF model, so it has one single optimization step through a rating prediction.

5.5.2 Comparison between Neighbors Selection Methods

In this chapter, apart from the rating conversion methods, I also propose different neighbors selection methods to solve the problem of 'all friends are not equal'. With this assumption, each neighbor has its own influence on the target user, so it should be treated differently in the model. There are several ways of neighbors selection. In the existing neighborhood-based RSs, they select the neighbors to be used in their models in different ways. The brief comparison is described in Table 5.2.

Table 5.2: Comparison of neighbor selection methods among TransRS [73], AHCF [74], and other rating conversion methods

Methods	TrustAnt	TransRS	AHCF	Other
Number of neighbors	limited to the depth of search	fixed friend size	all friends	several
Type	ACO	distance-based	attention mechanism	cosine similarity

Number of Neighbors

In this thesis, I proposed three different neighbor selection methods. Although all of them share the same goal: to select high-quality neighbors, they perform their selection processes in different ways.

In TrustAnt [72], the model selects raters who have high pheromone levels. This selection process is based on the depth of search in the trust network. The more depth of search, the more raters can be found. However, this proposed neighbor selection does not guarantee the number of found raters. Some high pheromone paths might not contain rater. Therefore, the number of neighbors in this model depends on the depth of search.

In TransRS [73], the model selects the n best rater(s) to be neighbor(s) of the target user by computing the distances among users. This selection method is proposed based on the assumption that if two users have similar preferences, their embeddings should be close in the vector space. Therefore, the user who is the closest to the target user will be the best rater. Moreover, because this proposed model is user–based CF, it only selects the user's neighbors.

According to the number of neighbors, in TransRS [73], the number of neighbors n is one of the hyperparameters which need to be set manually in advance. I have to tune n to obtain the best set of neighbors which can consume a lot of resources. This also does not correspond to a real-world scenario where most users do not share the same number of friends.

While in AHCF [74], we can use all neighbors as an input, then the model will assign different weights based on similarity between each user and the target user. With this proposed friend selection method, we do not have to tune the n hyperparameter like in TransRS [73], which I believe is more practical in real-world situations.

Implementation

There are several ways of neighbor selection in CF RSs. The most common neighbors selection method in neighborhood-based CF is choosing neighbors based on cosine similarity.

On the other hand, my proposed methods are implemented in different ways. In TrustAnt [72], the neighbors selection method is proposed based on the pheromone in the trust network. The assumption is that the path that contains high pheromone should consist of high-quality users. Therefore, this selection method is implemented based on the ACO concept.

In TransRS [73], the neighbors selection method is proposed based on the assumption that if two users have a similar preference, the vector of them should be close in the vector space. Therefore, in this model, the user who is the closest to the target user will be the best rater. We can obtain the best n raters by calculating the distance between the target user and every user who has rated the target item. The user whose distance from the target user is minimum will be the best rater of the target user.

In contrast, AHCF [74] the neighbors selection module is based on using the attention mechanism. This attention mechanism can adaptively learn the similarity between each friend and the target user, and each historical item and the target item. With this module, users having the more attention scores will likely have more contribution in the prediction of the model.

Chapter 6

Incorporating Knowledge Graph Embedding to Recommender Systems

6.1 Introduction

The previous chapter describes how to solve problems in neighborhood-based CF. This chapter will focus on the details of how to solve problems in model-based CF.

In this thesis, since I would like to build an efficient hybrid CF model, all components of the model should be carefully considered. In model-based CF approach, how to model the relation between user and item is one of the important points to be considered. Collaborative metric learning (CML) [39], one of model-based CF approach, tried to construct user–item relationship by minimizing the distance between user–item in vector space, which lead to the geometric inflexibility problem because it tried to put user–item into the same point in the vector space. Inspired by Latent relational metric learning (LRML) [70], this work tries to alleviate the problem of geometric inflexibility by adopting the idea from TransE [12], the popular KG embedding technique. The KG concept is applied to construct the scoring function of the model and the optimization of the proposed hybrid model is also inspired by TransE.

Because the proposed TransRS [73] is not an end-to-end CF model, the KG embedding technique cannot be applied in the same way as in AHCF [74] does. Therefore, I tried to apply the KG embedding technique in a different way.

In order to propose the rating conversion method described in previous chapter 5.3.4, we need to know each user's rating patterns; in other words, user's preferences. There are many methods that can obtain rating patterns of each user. Recently, many researchers have tried to find novel and efficient ways [59, 82] to represent user's preferences. Some researchers, for

example, adopt the idea from the KG field, which is a translation-based embedding model, to represent users, items, and their relations in graph form [59, 82]. Therefore, in TransRS [73], KG embedding technique is applied in the early step before the rating conversion. This aims to obtain embedding vectors of users, items, and relations. The details of this method are described in the following subsection.

6.2 Translation-based Embedding Model for Rating Conversion in Recommender Systems

As mentioned in the previous subsection that TransRS [73] adopts KG embedding technique into the model in a different way from in AHCF [74]. This proposed model applies TransE [12] as the first step of the model. Applying TransE to the model can generate and represent the relation between a pair of user and item in vector form, instead of scalar form.

6.2.1 Embedding Vectors of Users and Items

Users and items are usually represented by vectors in a d-dimensional feature space R^d in RSs. Let U and I denote a set of users and items, respectively. For a user $u \in U$ and an item $i \in I$, I denote their feature vector as $\vec{v}_u \in R^d$ and $\vec{v}_i \in R^d$, respectively. The relations between an user-item pair are usually represented by rating score. In many RSs, the rating score is predicted by the inner product $\vec{v}_u \cdot \vec{v}_i$ of the user and item feature vectors with some additional terms.

In RSs, user's explicit feedback is given as a triple (u, i, s) meaning that a user u gives score s to an item i. Then, a set of explicit feedback forms a (bipartite) graph where nodes are users and items whereas edges are rating scores. This view leads to another type of model for users and items embedding. In the literature of knowledge graph, entities and relations compose a graph and both entities and relations are embedded into the same feature space.

By regarding users and items (resp. scores) as entities (resp. relation), we can make use of translation-based models which are often used in KG embedding. In the translation-based models, relations are represented by a feature vector as well. Since a vector has more expressive power than a scalar, we can expect to obtain higher quality feature vectors of users and items. Wang et al. proposed a translation-based model for recommender systems called Constrained Preference Embedding (CPE) [77]. I build a rating conversion model on CPE.

Embedding Model

Suppose a RS accepts ratings scores like $\{1,2,\cdots,S\}$. Then, let me introduce *rating score vectors* $\{\vec{r}_s \mid s = 1,2,\cdots,S\}$. In translation-based models, a triplet (u,i,s) is modeled as

$$\vec{v}_i = \vec{v}_u + \vec{r}_s \,. \tag{6.2.1}$$

When embedding users and items, the model wants to locate a user u and an item i close to each other in the feature space if u likes i. Higher score means the user likes the item more. Therefore, I force the following constraint on the rating score vectors: for any rating score p and q

$$\|\vec{r}_p\|_2 < \|\vec{r}_q\|_2 \;\; \mathrm{if} \;\; p > q \tag{6.2.2}$$

where $\|\cdot\|_2$ stands for the L_2 norm.

Loss Function

In order to learn the model, let me derive the loss function. For a triple (u,i,s), error of the embedding is given as

$$s(u,i,s) = \|\vec{v}_u + \vec{r}_s - \vec{v}_i\|_{L_2} \tag{6.2.3}$$

from Eq. (6.2.1).

In order to handle constraints Eq. (6.2.2), I introduce the following quantity as in [77]:

$$C \equiv \sum_{p>q} \frac{w_{p,q}}{e} \ln g(\|\vec{r}_q\|_2^2 - \|\vec{r}_p\|_2^2) \tag{6.2.4}$$

where p and q are rating scores; e is a hyperparameter; $w_{p,q}$ is the weight of the preference comparison between p and q computed by the product between the frequency in the training triples of p and q; $g(\cdot)$ is a monotonic function.

Unlike CPE, I adopt the margin-based loss as the loss function to learn embeddings of the users, items, and rating scores. For each triple (u,i,s) in training data Δ, I sample (u',i',s') which is a corrupted pair (i.e., negative sampling). Finally, the loss function is given as

$$L \equiv \sum_{(u,i,s)\in\Delta} \sum_{(u',i',s')\notin\Delta_{u,i,s}} max(0, s(u,i,s) + \gamma - s(u',i',s') - C), \tag{6.2.5}$$

where $\Delta_{u,i,s}$ is negative samples by corrupting (u,i,s) with replacing u and i with user and item as in [12]. The symbol γ is the margin that separates the positive and negative samples.

We can obtain the embedding user vectors $\{\vec{v}_u^*\}_{u\in U}$, item vectors $\{\vec{v}_i^*\}_{i\in I}$, and rating score vectors $\{\vec{r}_s^*\}_{1\le s\le S}$ by minimizing the loss function Eq. (6.2.5).

6.3 Attentive Hybrid Collaborative Filtering

This subsection explains the details of the final layer of AHCF [74]. This model is an end-to-end NN. Inspired by LRML [70], it learns latent relation vectors for user–item pairs by adopting an idea from TransE [12] to model the latent relation vectors. This mitigates the geometric inflexibility of CML [39], whose objective function aims to place user–item pairs at the same point by minimizing the distance between each user–item interaction. Therefore, this proposed model learns the latent relation vector between a user–item pair as follows.

6.3.1 The Latent Relation Modeling

In RSs, there are several options to represent or handle the relation between user and item. The first one is *binary relation*, which represents relation in the terms of 'like' and 'dislike' or 'unknown'. Another common way to represent relation is a concrete score in scalar form. In this thesis, I represent the relation between user and item with more complexity than the binary and scalar score. I propose to represent the relation between user and item in this model with the *latent relation vector*, which represents their relation in multi-dimensional vector space. This latent relation vector is inspired by TransE [12].

For each pair comprising a user u and an item v, the scoring function $s(u,v)$ is defined as

$$s(u,v) = \|\mathbf{u}+\mathbf{r}-\mathbf{v}\|_2^2, \tag{6.3.1}$$

where r is the latent relation vector obtained from Eq. (5.4.15) and $\|\cdot\|_2^2$ is the L2 norm of the vector $\mathbf{u}+\mathbf{r}-\mathbf{v}$. This equation is adopted from TransE [12] because TransE is the simplest and most understandable KG embedding technique.

6.3.2 Objective Function

Like TransE [12], I also use the pairwise ranking loss or *hinge loss* for optimization. For each positive (observed) user–item pair $\langle u,v\rangle$, the model performs negative sampling by sampling a corrupted user–item pair denoted as $\langle u',v'\rangle$. The corrupted user–item pair goes through the same user–item embedding layer. The pairwise hinge loss is then defined as

$$L = \sum_{(u,v)\in\Delta} \sum_{(u',v')\notin\Delta} max(0, s(u,v)+\Lambda-s(u',v')), \tag{6.3.2}$$

where Δ is the set of all positive user–item pairs and Λ is the margin that separates the observed pairs and the corrupted samples.

Table 6.1: Comparison between applying KG to TransRS [73] and AHCF [74]

Methods	TransRS	AHCF
Purpose	for learning pre-trained embeddings	for optimization
Relation vector	shared relation vectors	specific relation vectors

6.4 Discussion

According to the previous subsections 6.2.1 and 6.3.2 that respectively demonstrate how KG embedding technique are adopted and applied in TransRS [73] and AHCF [74], this subsection will discuss on the similar and different points between these two approaches. The brief comparison is described in Table 6.1.

6.4.1 Purpose of Applying Knowledge Graph Embedding technique to Recommender Systems

The purpose of applying the KG embedding technique to TransRS [73] and AHCF [74] are different. In TransRS [73], TransE [12] technique with modification is applied to the model to generate the approximate users, items, and relations vector. In other words, I modified the traditional TransE KG embedding technique based on the assumption that 'if user u rate item i with a high score, vectors of user u and item i should be placed close to each other in the vector space.'. By adding constraint Eq. 6.2.2 to TransE model, the output from this step are approximately users, items, and relations vectors.

On the other hand, in AHCF [74], I utilize the idea of TransE [12] to model the latent relation between user–item pair and mitigate their learning process in the optimization. In this work, I did not directly utilize the traditional TransE model to generate approximately users, items, and relations vectors, but only used the idea of how the TransE model relationship between user and item. With this $\mathbf{u} + \mathbf{r} - \mathbf{v} = 0$, I define the scoring function of this model as in Eq. 6.3.1.

6.4.2 Relation Vector

Apart from the purpose of applying KG embedding to each model, the relation vectors generated in each model are also different. In TransRS [73], the relation vectors are obtained from the modified TransE model, which is the first step of TransRS. The relation vectors in TransRS refers to rating score vectors r, which is one of the model parameters in the first

module of TransRS. These rating score vectors are shared among users and items in the vector space. According to TransE's properties, the relation vectors are shared among entities in the vector space. Therefore, it can handle only a one-to-one relation, not suitable for one-to-many, many-to-one, and many-to-many relations. This kind of relation vector learning might lead to low accuracy prediction because all entities that share the same relation are forced to be close to each other in the vector space.

On the other hand, in AHCF [74], I adopt only the idea of TransE to model the latent relation vector between a pair of user–item. This proposed latent relation vector is a unique and specific relation vector between a pair of user–item unlike in TransE which relation vectors are shared among entities. With this proposed latent relation vector, it can handle one-to-many, many-to-one, and many-to-many relations better than in TransRS [73].

Chapter 7

Integration and Recommendation

7.1 Introduction

In previous chapters, I present three main components of this thesis, which are

- Utilizing neighbors into a hybrid CF model (Chapter 4),
- Neighbors selection and Rating Conversion in a hybrid CF (Chapter 5), and
- Incorporating Knowledge Graph Embedding to Recommender Systems (Chapter 6).

In order to build a hybrid CF model, this chapter demonstrates how these components are combined together for making recommendations. This chapter provides a high-level explanation on my proposed models, which are 1) Integrating the importance levels of friends into trust-based ant-colony recommender systems (TrustAnt) [72] in section 7.2, 2) Translation-based Embedding model for Rating Conversion in Recommender Systems (TransRS) [73] in section 7.3 and 3) Attentive Hybrid Collaborative Filtering for Rating Conversion in Recommender Systems (AHCF) [74] in section 7.4.

7.2 Trust-based Ant Recommender System

TrustAnt [72] consists of the following two phases:

Learning phase

- Trust network and input representation (see Section 5.2.1)
- Proposed ant algorithm (see Section 5.2.2)

Prediction phase

- Rating conversion and weight adjustment (see Section 5.2.3)
- Rating prediction

In the previous chapters, the details of each step in the learning phase. In this subsection, I will describe how to do the rating prediction in this model.

7.2.1 Prediction value

After obtaining the final rating W' (Eq. 5.2.10) of all selected raters from the best paths, now the model has suitable ratings from every rater using the same perspective as the target user. If the weight is a constant value, the model use these ratings to calculate the predicted rating for the target item i by the target user A as:

$$p_{Ai} = \frac{\sum_{u \in N_A} W'_{u \to A}(r_{ui})}{n(N_A)}, \tag{7.2.1}$$

where $n(N_A)$ is the number of selected raters from the global pheromone-update step.

As mentioned in the previous subsection, this model has two ways to calculate the weight. If the model obtains the weight from Eq. 5.2.11, the predicted rating for the target item i by the target user A is calculated as follows:

$$p_{Ai} = \frac{\sum_{u=1}^{N} W'_{u \to A}(r_{ui})}{\sum_{u=1}^{N} \gamma_u} \tag{7.2.2}$$

where N is the number of selected raters from the global pheromone-update step.

For example, assume that Figure 7.1 (a) shows the best path from the target user A; it contains four users. In this path, only users u_2 and u_4 have ratings for the target item i. Even if users u_2 and u_4 both gave scores of five to item i, their scores should not be treated equally. The first step is to transpose each rater's rating to the target user's perspective by using Eq. 3.3.3. Assume that, after transposing, the rating of five by user u_2 is transposed to 4.1 while that by user u_4 is transposed to 3.8, as shown in Figure 7.1 (b).

The second step is to adjust the rater's weight by using Eq. 5.2.10. To achieve this, the transposed rating of user u_2 must be multiplied by the weight of user u_2. I will show the calculations for both constant weight and trust-based weight.

For a constant weight, the transposed rating of user u_2 must first be multiplied by 0.9 and that of user u_4 must be multiplied by 0.7, giving ratings of 3.69 and 2.66. The model then calculates the prediction value via Eq. 7.2.1, and the result is 3.175. This means that target user A is predicted to rate the target item i with a score of 3.175.

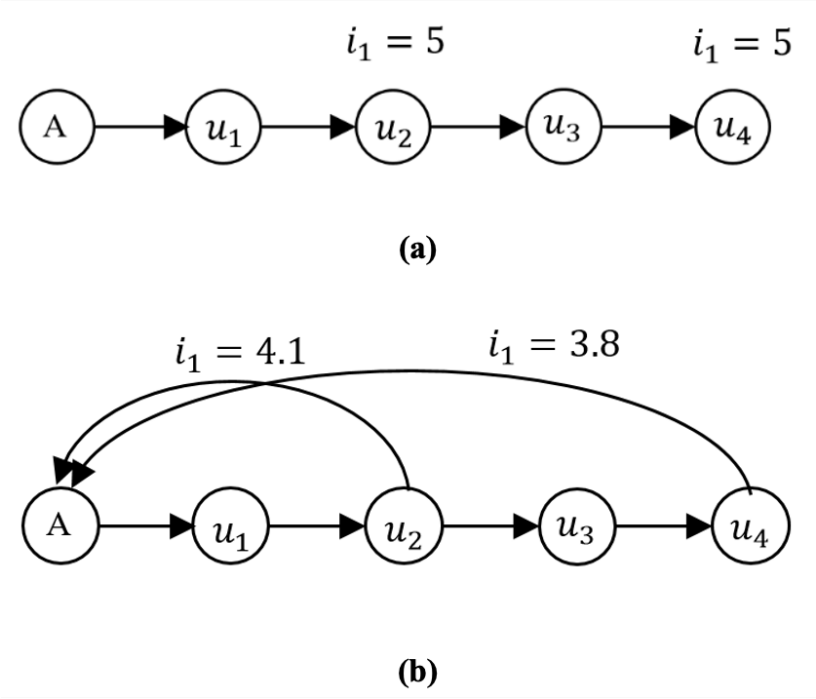

Figure 7.1: Examples of the prediction calculation

For a trust-based weight, the transposed rating of user u_2 must be multiplied by the weight that can be calculated by Eq. 5.2.11 and similarly for user u_4. Assume that the weights for user u_2 and user u_4 are 0.8 and 0.7, respectively. After this step, we have ratings of 3.28 and 2.66. The model then calculates the prediction value via Eq. 7.2.2, and the result is 3.96.

7.2.2 Experimental Evaluation

In this subsection, the details of experiments applying ACO concept to the trust-based RSs are described. For evaluation, I compare the experimental result with ALT-BAR [10], which also uses the ACS algorithm.

Dataset

The Epinions dataset was used in this work. It contains 49,290 users who have rated 139,738 unique items at least once. The rating range is 1–5 (5 being the best). The dataset contains 664,824 reviews and there are 487,181 explicit trust statements among the users. The data are separated into two categories, trust data, and rating data.

The data are very sparse. Most users rate just a few items when compared with the whole dataset. Some users have not rated anything, and some items are not rated by anyone. I

removed these users and items that have no rating or have fewer than 10 rating records. The numbers were therefore reduced to 36,363 users and 11,657 items.

In the experiment, I use both the raw dataset and the preprocessed dataset.

Evaluation Metrics

To evaluate this proposed model and compare the results with those obtained with ALT-BAR [10], I used the mean absolute error (MAE) and rating coverage (RC) as the evaluation metrics.

Mean Absolute Error (MAE) measures the closeness of the predictions to the eventual outcomes. It is calculated by averaging the absolute difference between the expected rating and actual rating for the target item, as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{r}_i - r_i| \tag{7.2.3}$$

where n denotes the number of predicted ratings, $\hat{r}_i$ is the predicted rating for target item i, and r_i is the user's actual rating for the target item i.

Rating Coverage (RC) measures the ability of the RS to predict a rating for a target item, as follows:

$$Coverage = \frac{n^{predict}}{n^{actual}} \times 100\% \tag{7.2.4}$$

where $n^{predict}$ refers to the number of predicted ratings and n^{actual} refers to the number of actual ratings.

Experimental Results

Proposed method vs ALT-BAR

First, I compare the results obtained from running the proposed method on the original Epinions dataset (not pre-processed) against ALT-BAR.

Because the number of users in the dataset was very large, the network is very large. Because it was difficult to generate recommendations for the full network, I selected 500 users who had each rated more than 10 items to ensure that the system would have a reasonable chance of finding raters.

As mentioned in the previous subsection, there are two ways to adjust weights for each rater. In this experiment, I used the first weight adjustment algorithm, which adjusts by a constant value. I used the "leave-one-out" technique to test the ability of the proposed method to predict items' ratings. I assigned parameters to each experiment as follows:

Table 7.1: MAE and RC of the proposed method and ALT-BAR for 500 users

Evaluation metric	Algorithm	
	Proposed method	ALT-BAR
MAE	**0.99**	1.02
RC	**70.8%**	28.8%

- number of ants: $k = 10$
- number of iterations: $t = 10$
- search depth: $d = 10$.

The ALT-BAR is implemented by assigning parameters as follows:

- number of ants: $k = 10$
- number of iterations: $t = 10$
- search depth: $d = 10$
- path trust threshold: 0.2.

Table 7.1 shows that the proposed method generates more accurate predictions than ALT-BAR, because the MAE for the proposed method is less than that for ALT-BAR. Table 1 also shows that the proposed method provides more coverage than ALT-BAR. From the experiment with 500 users, the coverage for the proposed method's prediction is 354 users or 70.8%, whereas the coverage for ALT-BAR's prediction is only 144 users or 28.8%. This demonstrates that my proposed method provides more coverage for predictions.

Apart from the comparison between the proposed method and ALT-BAR, I wanted to test the effects of the proposed weighting methods on the prediction step. I conducted the following experiments.

Weight adjustment

I compared the effects of the two weighting methods. $W1$ uses constant weights as described in Section 5.2.3, and $W2$ uses weights obtained using Eq. 5.2.11. I also tested a system with no weight adjustment. I set the parameters as follows:

- number of ants: $k = 10$

Table 7.2: MAE of experiments on different weight adjustment methods for 500 users

Experiment	Algorithm		
	W1	W2	No weight
d = 10	0.99	**0.96**	1.51
d = 20	1.02	**1.00**	1.75

- number of iterations: $t = 5$
- search depth: $d = 10, 20$.

This experiment was run on the preprocessed Epinions dataset described in the previous subsection. I selected the same 500 users as in the previous experiment. To test the effects of weight adjustment, this experiment used the raw ratings (not transposed ratings) to generate the recommendations.

Table 7.2 shows that more accurate predictions are generated by $W2$, which is the trust weight. This experiment shows that using trust weights is better than using constant weights. I also showed that the system using both weight $W1$ and $W2$ in the prediction step provides more accurate predictions than the system without weights.

Transposed ratings

Apart from the weight adjustment in the prediction step, I also proposed transposing a rater's rating to match the active user's perspective. Therefore, in this subsection, I tested the effects of my proposed transposed rating and compared it with the raw rating. I compared the experimental results between a system using transposed ratings (M1) and one that uses raw ratings (M2) for different depths of search. Note that, because this experiment tests the effects of transposed ratings, I did not use the weight adjustment method. M1's results are calculated by averaging the transposed ratings while M2's results are calculated by averaging the raw ratings. The transposed ratings can be calculated from both real ratings and latent ratings. The system will use latent ratings when there are no real ratings to use in the calculation. Because of this, I compared the predictive ability of real and latent ratings by separating the experiment into two parts. I used the same parameters as in the experiment in previous subsection. The results are shown in Table 7.3.

Table 7.3 shows that when real ratings are used in the transposing step, system M1 provides more accurate predictions than M2 for both $d = 10$ and $d = 20$. Thus, using transposed ratings calculated from real ratings to generate the predicted ratings, provides more accurate

Table 7.3: MAE of experiments using transposed ratings and raw ratings in the prediction step for 500 users

Method	Real ratings in transposing step		Latent ratings in transposing step	
	M1	M2	M1	M2
d = 10	1.45	1.51	1.09	1.02
d = 20	1.64	1.75	1.12	1.03

Table 7.4: MAE of combination experiments for 500 users

Method	Real ratings in transposing step		Latent ratings in transposing step	
	M1	M2	M1	M2
d = 10	0.90	1.45	1.07	1.09
d = 20	0.95	1.64	1.15	1.12

recommendations than the traditional method. However, the experimental results in Table 7.3 show that when latent ratings are used in the transposing step, system M2 provides better accuracy than M1 for both $d = 10$ and $d = 20$. It can be concluded that the latent ratings obtained from Singular Value Decomposition (SVD) are not efficient enough, so using those latent ratings in the transposing step may provide inefficient transposed ratings. I will discuss this issue further in the discussion section.

Combination of Transposed Ratings and Weight Adjustment

The experimental results in the previous subsections have shown that the proposed weight adjustment and transposed ratings can provide more accuracy than the traditional method. Therefore, in this subsection, I combine these two proposed methods. Because the results in Table 7.2 show that trust weights provide higher accuracy, I used these trust weights in this experiment.

This experiment will be separated into two parts, using real and latent ratings in the transposing step. The parameters are the same as in the experiments above.

System M1 combines transposed ratings with trust weights in the prediction step, while system M2 uses transposed ratings and averages them. The experimental results are shown in Table 7.4.

Table 7.4 shows that with real ratings in the transposing step, M1 provides more accurate predictions than M2 for both $d = 10$ and $d = 20$. It can be concluded that using transposed

Table 7.5: Average RC of all experiments for 500 users

Experiment	Rating Coverage (RC)
d = 10	77.2%
d = 20	83.2%

ratings calculated from real ratings by applying weight adjustment to each rater, provides more accurate predictions than transposed ratings without weight adjustment. Although the results for M1 using real ratings in the transposing step are better than those for M2 for both $d = 10$ and $d = 20$, the results for M1 using latent ratings in the transposing step are less clear. From Table 7.4, with latent ratings in the transposing step, M1 provides more accurate recommendations than M2 only for $d = 10$, while M2 provides more accurate recommendation than M1 for $d = 20$. It can be concluded that there may be some raters located very far away from the active user who will have very small weights. When a very small weight combines with a transposed rating, the final predicted rating may not be accurate enough. This issue will be discussed further in the next section.

Rating Coverage

Apart from the accuracy level of all experiments in the previous subsections, I also measured the RC for each experiment. The results for all experiments are shown in Table 7.5.

Table 7.5 shows that the experiments with $d = 10$ and $d = 20$ provide more coverage than those shown in Table 7.2. From the experiments with 500 users, the coverage for $d = 10$ is 386 users or 77.2%, and the coverage for $d = 20$ is 416 users or 83.2%, whereas the coverage from Table 7.2 ($d = 10$) is only 354 users or 70.8%. This demonstrates that the use of the preprocessed dataset provides more coverage for predictions.

7.2.3 Discussion

In the previous subsection, I showed that for the Epinions dataset the proposed method can provide more accurate predictions with better coverage than can ALT-BAR [10], and that transposing the raters' ratings into the active user's perspective and adjusting the weight of each rater can improve the accuracy of predictions.

In previous experiments with ALT-BAR, the results were compared with two other algorithms, traditional CF and (MoleTrust) MT [56]. For the Epinions dataset, those results demonstrated that ALT-BAR was more accurate and provided more coverage than either CF

or MT. Thus, my proposed method can be expected to perform better than both CF and MT in terms of both accuracy and coverage.

I now discuss the two main contributions: the improved method for selecting high-quality raters, and the improved prediction step.

Selecting High-quality Raters

To achieve our aim of selecting high-quality raters, the model performed preprocessing of the dataset and improved the global pheromone-update function. This contribution enables the system to provide better accuracy and coverage. I also varied the depth of search d to test its influence on the accuracy level or coverage level.

This contribution has enabled the proposed method to obtain greater accuracy and coverage than ALT-BAR.

Trust Generation

Trust values are inputs to the system. If the trust values can be suitably represented, good outputs can be expected. Most datasets, including the one I used, are sparse. I, therefore, performed a preprocessing step to obtain better input representation. The experimental results in the previous subsection show that using the preprocessed dataset provides better coverage than using the original dataset.

I also proposed a method that generates trust values without using co-rated items and avoids the symmetry problem. This approach yields high-capability trust values, which lead to accurate recommendation results. Because ALT-BAR uses co-rated items to generate similarity and uses that similarity to generate trust values, it can incur sparsity problems.

Improvement of the Global Pheromone Update

I improved the process of updating pheromone levels by using an additional factor, the popularity of each user. In existing systems, only three factors are considered: the pheromone level on each edge, the distance between each user and the active user, and the number of items rated by a user that are unrated by the active user. My additional factor raises the pheromone level for reliable users and helps capture higher–quality raters. These improvements led to better coverage.

Moreover, I considered the number of raters on a selected path whereas ALT-BAR does not. I improved the optimization condition to update global pheromone levels by updating only paths that contain raters. This increases the likelihood that a path containing raters will have higher pheromone levels than those without raters. Paths that contain more raters will be more likely to be selected to generate recommendations. Therefore, higher prediction coverage will be obtained.

The depth of search d is another important parameter in finding more raters. The system can find more raters with higher values of d: the results have shown that $d = 20$ can find more raters than $d = 10$. However, in terms of accuracy, higher values of d result in higher values of MAE. Because of the weight adjustment in the second contribution, raters who are located very far from the active user will be assigned very small weights. As their assigned weight is very small, those raters are too far away to use because they are not high–quality raters for the active user. Using their ratings may reduce the accuracy of the predicted ratings for the active user. It can be concluded that there are trade-offs between coverage and accuracy. Therefore, using a larger depth of search may provide more coverage, but may also obtain low-quality raters.

Improving the Prediction Step

The second goal of this work is to improve the prediction step. In this contribution, I tried to reduce the improper rating-range problem by applying a rating-pattern conversion to transposed ratings to match the active user's perspective. Moreover, to obtain more accurate results, I considered the influence level of each rater based on the distance between the rater and the active user. This is called weight adjustment.

The experimental results show that this contribution helped the proposed method to provide more accurate recommendations.

Transposed Ratings

First, I transposed a rater's rating to match the active user's perspective instead of using the raw rating, as ALT-BAR does. The rating value used to generate the recommendation must be placed within the same range as that used by the active user before it is used.

The experimental results in Table 7.3 show that the accuracy of the system using transposed ratings is only higher than that of the system using raw ratings when real ratings are used in the transposing step. However, from Table 7.3, if the system uses latent ratings in the transposing step, the non-transposed ratings provide more accurate predictions than the transposed ratings for both $d = 10$ and $d = 20$. It can be concluded that the latent ratings obtained from SVD are not efficient enough for use in the transposing step. Therefore, when inefficient latent ratings are used to generate the transposed ratings, the resulting transposed ratings may be low quality and inefficient, making the predicted rating also low quality.

Weight Adjustment

As well as transposing the ratings, I also considered the influence level (importance level) for each rater as a weight calculated from the distance between the rater and the active user. This work studies two different weight adjustment methods: constant weights and the trust weights.

The experimental results in Table 7.2 show that the system applying weight adjustment provides more accurate predictions than the system without weight adjustment. Moreover, the results show that trust weight is better than constant weight. It can be concluded that the distance between the rater and the active user influences the accuracy level of the recommendations.

Combination of Transposed Ratings and Weight Adjustment

I also studied the combination of transposed ratings and weight adjustment to test the ability of my proposed method against the traditional method. The experimental results of the proposed combined method are shown in the previous subsection. Table 7.4 shows that the combination of transposed ratings and trust weight adjustment can provide more accurate results than using either method alone when the transposed ratings are calculated from the real rating.

On the other hand, the combined method is less successful when latent ratings are used to calculate the transposed ratings. The results show that if $d = 20$, the combined method provides less accurate predictions than transposed ratings with no weight adjustment. This suggests the following two points.

1. Latent ratings are not real ratings, so they may not be appropriate in the transposing step. When inappropriate ratings are used, the resulting transposed ratings may have low quality and may not be good enough to use for predictions.

2. Some raters may be located too far away from the active user. Those raters will be assigned very small trust weights and combined with transposed ratings, may cause the final predicted rating to be of low quality.

Thus, if inappropriate latent ratings are combined with inappropriate weight adjustment, the predicted rating will also have an improper value.

Therefore, with my proposed method, the best recommendations will be obtained if we use the combination of transposed ratings from real ratings, trust weight adjustment, and appropriate depth of search.

7.2.4 Conclusion

I have extended previous work of applying the ACO concept to trust-based RSs [71], and aimed to improve its accuracy and coverage compared with our prior work and other existing systems.

My proposed method makes two main contributions: 1) it finds higher-quality raters, and 2) it improves the prediction step. With the first of these, we can find high-quality raters

via a new trust-value calculation and a preprocessed dataset. I also improved the global pheromone-update process by considering four factors: the pheromone level on each edge, the distance between each user and the active user, the number of items rated by each user that are unrated by the active user, and the popularity of each user. These four factors improve the rating process. I also proposed a new optimization condition for updating global pheromone levels by updating only paths that contain raters. This proposed method enables the system to find more and better raters.

My second contribution is to improve the prediction step by transposing user ratings to match the active user's perspective and assigning a weight for each rater with respect to the active user before calculating the predictions for the active user. I proposed a constant weight and a trust weight generated from the trust value of the rater and the distance between the rater and the active user. With this proposed method, the system could generate more accurate recommendations and provide a higher RC.

The experimental results and comparisons have demonstrated that my proposed method can provide more accurate and wider-coverage predictions than the previously proposed method as well as existing trust-based ant RSs. Moreover, the combination of transposed ratings with trust weight adjustment and appropriate depth of search d in the prediction step can provide more accuracy and better RC.

7.3 Translation-based Embedding model for Rating Conversion in Recommender Systems

This proposed model is built by the following three main steps, which are:

1. **Embedding vectors of users and items**: extracts the rating pattern of each user and generates the approximate rating vector of each user. The detail of this step is described in subsection 6.2.1.

2. **Rating score conversion**: converts rater's rating score into target user's perspective.

3. **Rating prediction**: predicts the score that the target user will give to the target item.

Figure 7.2 shows the components and flow of the model. As this work is a building boxes model, the output of the previous step is used to be an input of the next step. The details of the first two steps are already described in chapter 5 and 6. This section will then focus on the rating prediction part of the model.

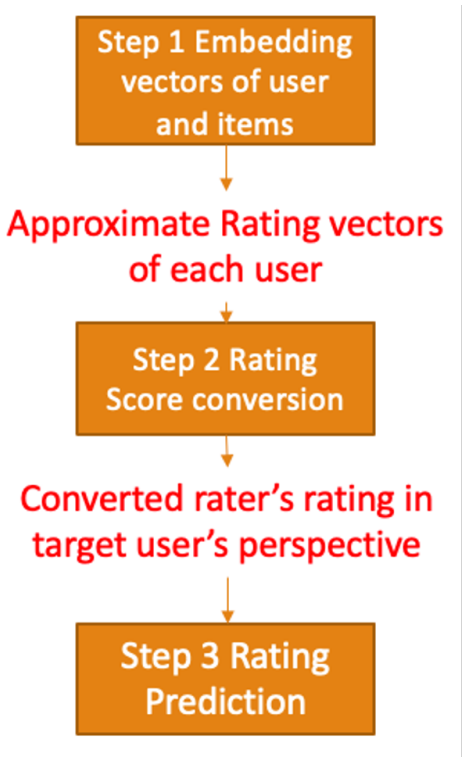

Figure 7.2: A workflow architecture of TransRS

7.3.1 Rating Prediction

After obtaining the best rater's converted rating vector by Eq. (5.3.6), I use these converted rating vectors to calculate the predicted rating that the target user should give to the target item. In this step, the idea of CPE is applied to the model by transforming a rating vector to the preference score. CPE proposed user u's preference on item i as

$$p_{u,i} = \frac{1}{\|\vec{v}_i^* - \vec{v}_u^*\|_2^2 + 1}. \tag{7.3.1}$$

For this proposed method, I include n best rater's information to predict the preference of user u on item i. This proposed preference equation is defined as follows:

$$p_{u,i} = \rho \cdot \frac{1}{\|\vec{v}_i^* - \vec{v}_u^*\|_2^2 + 1} + (1-\rho) \cdot \frac{1}{\|\vec{r}_{bf(u,i),i,u}\|_2^2 + 1} \tag{7.3.2}$$

where the rating score used in the second term is the converted best rater's rating vector given by Eq. (5.3.6), whereas ρ is the ratio of the linear combination of two score predictions.

Finally, I normalize the obtained preference score into k range score based on each dataset and this is the predicted rating that the target user u will give to the target item i.

Table 7.6: Dataset summary

Dataset	Ratings	Users	Item
FilmTrust	35,497	1,508	1,036
MovieLens100K	100,000	942	1,682
MovieLens1M	1,000,209	6,040	3,706

In case the number of rater n is more than 1, I do the step explained in 4.2 on each rater. Then, in step 7.3.1, I calculate the predicted rating of each rater and find the average of predicted ratings from those n raters.

7.3.2 Experimental Evaluation

To evaluate the ability of n best raters and rating conversion method, in this subsection, I present details of the experiment applying rating conversion to a translation-based embedding model in RSs. I compare the experimental results with CPE [77] which also uses translation-based embedding model concept in RSs. Also, I compare it with Matrix Factorization (MF) [49] which is a state-of-the-art method in RSs.

Dataset

I evaluate the method using three real-world datasets, which are FilmTrust[1] and MovieLens[2] (100K and 1M). FilmTrust is a recommendation dataset crawled from the entire FilmTrust website. MovieLens is a widely adopted benchmark dataset for CF in the application domain of recommending movies to users. In this work, I use two configurations of MovieLens, which are MovieLens100k and MovieLens1M. The numbers of users and items of each dataset are listed in Table 7.6.

Implementation and Parameter Setting

In this model, I have modified the source code of OpenKE [35], an open-source toolkit which provides a unified framework and various fundamental models to embed KGs into continuous low-dimensional spaces.

[1] https://www.librec.net/datasets.html
[2] https://grouplens.org/datasets/movielens

Each dataset is separated into three parts; 60% is used for training, 20% is used for validation and the remaining 20% is left for testing. All models are trained until convergence. I optimized all the models by using stochastic gradient descent (SGD) method.

For all the approaches, the embedding dimension d is tuned amongst $\{10, 20, 50\}$; the learning rate is tuned amongst $\{0.01, 0.001.0.0001\}$; the margin γ is tuned amongst $\{0.25, 0.5, 1.0, 2.0\}$. For the FilmTrust dataset, batch size is tuned amongst $\{10, 20, 50\}$. For the MovieLens100k dataset, batch size is tuned amongst $\{120, 240, 360, 480\}$. For MovieLens1M, batch size is tuned amongst $\{120, 480, 1200, 3600\}$. I set e and w_{f_p,f_q} in Eq. (6.2.4) to 1 as in CPE.

Evaluation Metrics

In this experiment, I perform the evaluation based on two tasks: *rating prediction* and *ranking recommendation*. To evaluate this proposed model and compare the results with the baseline methods, I used the mean absolute error (MAE) and root mean square error (RMSE) as the evaluation metrics for the rating prediction task. For the ranking recommendation task, I used Precision@k and Recall@k.

Although plenty of studies have shown that the rating prediction task poorly correlates with the recommender precisions, I would like to show that this proposed method can provide better accuracy than the other baselines in both ranking recommendation and rating prediction tasks.

Experimental Results

The objective of this experiment is to test the ability of the rater and rating conversion method. For this purpose, I compared the experimental results with CPE [77]. First, I divided the experiments into three main groups:

- **CPE**: like [77], I only used the target user's information (approximate rating vectors) to compute the predicted ratings on target items.
- ***n* best rater(s)**: since I believe that the rater in the system may have an influence on the target user, so in the group, I used the rater's information to compute the predicted ratings on target items. I refer to this group as nBR.
- **CPE with *n* best rater(s)**: I combined CPE and n best rater(s). I refer to this group as CPE + nBR.

Note that n is the number of best raters. In this experiment I set n to 1, 5 and 10.

To test the ability of rating conversion, I conducted three methods on each group, which are:

- **No rating conversion**: using obtained rating from Eq. 7.3.1 without any rating conversion like in CPE.
- **Normalization**: basic normalization method

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{7.3.3}$$

  where x is the target rating, x_{min} and x_{max} are the minimum rating and maximum rating of that user respectively.
- **Proposed rating conversion method**: unlike the basic normalization method, this experiment uses rating conversion in Eq. 7.3.2.

Note that normalization and this proposed rating conversion method cannot be applied to CPE group because there are no rater's rating to normalize or to convert.

Rating Prediction

There are two common evaluation tasks in RSs, which are rating prediction and ranking recommendation. In this work, both evaluation tasks are done. First, for the rating prediction, I predict the score that the target user should give to the target item and use MAE and RMSE as the evaluation metric. The experimental results are shown in Table 7.7.

From Table 7.7, for FilmTrust dataset, CPE + nBR with the proposed rating conversion provides the highest accuracy among the others. For MovieLens100K and MovieLens1M, n best rater(s) with the proposed rating conversion can provide the highest accuracy.

Moreover, the experimental results showed that just normalization is not enough. Applying the basic normalization method provides worse results than applying my proposed rating conversion method on all datasets.

From the above mentioned, I intend to show the ability of the target user's best rater and my proposed rating conversion method. The experimental results have shown that target user's raters have an influence on target user. Including the best raters of the target user to generate the recommendation can make more accurate predictions than CPE.

According to the number of the best rater n, in Table 7.7, I choose the parameter n that performs the best result. For FilmTrust, $n = 5$ provides the best result. For MovieLens100K and MovieLens1M, $n = 10$ provides the best result.

Ranking Recommendation

Table 7.7: MAE and RMSE of experiments on different rating prediction method

Dataset	Method	CPE		*n*BR		CPE + *n*BR	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
FilmTrust	No rating conversion	0.7873	0.9532	0.7861	0.9443	0.7855	0.9446
	Normalization	-	-	1.1658	1.4188	0.9476	1.1319
	Proposed method	-	-	0.8109	1.0475	**0.7491**	**0.9196**
ML100K	No rating conversion	1.1176	1.4236	0.9709	1.1824	0.9818	1.1905
	Normalization	-	-	1.2188	1.5013	1.0893	1.3294
	Proposed method	-	-	**0.9684**	**1.1614**	0.9822	1.1791
ML1M	No rating conversion	1.0792	1.3241	1.0738	1.3187	1.0761	1.3208
	Normalization	-	-	1.3802	1.6939	1.2112	1.4816
	Proposed method	-	-	**0.9664**	**1.1522**	1.0106	1.2094

Apart from rating prediction, the ranking recommendation is done in this work. For ranking recommendation, I ranked the items according to the predicted scores of the item in the test data. Then, the top k items with the greatest predicted rating score are recommended to the target user. I used precision@k and recall@k to evaluate this task. They can be computed by the followings:

$$Precision@k = \frac{|\{Recommended\ items\ that\ are\ relevant\}|}{|\{Recommended\ items\}|} \tag{7.3.4}$$

$$Recall@k = \frac{|\{Recommended\ items\ that\ are\ relevant\}|}{|\{Relevant\ items\}|} \tag{7.3.5}$$

An item is considered relevant if its actual rating is equal to or greater than a given threshold. An item is considered to be recommended if its predicted rating is equal to or greater than the threshold, and it is in the top k items.

In this experiment, I use 3 and 3.5 as the threshold of FilmTrust, and MovieLens, respectively, and set k to 10. The experimental results are shown in Table 7.8.

For all datasets, this proposed rating conversion method on n best raters provides better results than CPE only and CPE with n best raters in terms of both Precision@10 and Recall@10.

For most datasets and baselines, I found that the learning rate = 0.001 works well. For the embedding dimension d, I found that in FilmTrust and MovieLens100K, d = 10 works well.

Table 7.8: Precision@10 and Recall@10 of experiments on different method

Dataset	Method	CPE		*n*BR		CPE + *n*BR	
		P@10	Re@10	P@10	Re@10	P@10	Re@10
FilmTrust	No rating conversion	0.6791	0.5642	0.5162	0.3697	0.5281	0.3921
	Normalization	-	-	0.5515	0.3027	0.5185	0.4283
	Proposed method	-	-	**0.6904**	**0.6382**	0.6893	0.6371
ML100K	No rating conversion	0.5576	0.6274	0.5630	0.6434	0.5621	0.6385
	Normalization	-	-	0.5148	0.4261	0.5321	0.4900
	Proposed method	-	-	**0.5939**	**0.6628**	0.5652	0.6403
ML1M	No rating conversion	0.5176	0.4335	0.5185	0.4283	0.5177	0.4306
	Normalization	-	-	0.6088	0.3336	0.4845	0.3163
	Proposed method	-	-	**0.6734**	**0.5881**	0.5412	0.5236

However, in MovieLens1M, $d = 50$ works well. For the margin γ, I found that it depends on datasets; γ= 1.0 works well in FilmTrust, $\gamma = 0.25$ works well in MovieLens100K, and γ= 0.5 works well in MovieLens1M. For the batch size B, I found that it depends on the size of the dataset as well; $B = 10$ works well in FilmTrust, $B = 360$ works well in MovieLens100K, and $B = 1200$ works well in MovieLens1M.

Comparison with Matrix Factorization

Apart from comparing this proposed method in TransRS [73] with CPE and another rating conversion method, I also compared this proposed model with Matrix Factorization (MF) [49] which is the state-of-the-art method in RSs field. MF is a standard baseline for CF that models the relationship between user and item by inner products. In this experiment, I used MF-based algorithms in the Surprise library [3]. The MF-based algorithm that I chose is biased SVD. The parameters are all set as default as in the Surprise library. The experimental results are shown in Table 7.9.

In Table 7.9, I choose the results that provide the best performance on each dataset from Table 7.7 and 7.8 to compare with MF.

Although my proposed method cannot provide a better accuracy result than MF, it can be applicable with more datasets while MF cannot. This proposed model which adopts the idea of TransE [12], a translation-based embedding model in the KG field, can be applicable

[3] https://surprise.readthedocs.io

Table 7.9: MAE, RMSE, Precision@10 and Recall@10 of experiments on different method

Dataset	Method	MAE	RMSE	P@10	R@10
FilmTrust	MF	0.6369	0.7988	0.7630	0.7484
	*n*BR	0.7491	0.9196	0.6904	0.6382
ML100K	MF	0.7626	0.9586	0.6893	0.5134
	*n*BR	0.9684	1.1614	0.5939	0.6628
ML1M	MF	0.7022	0.8897	0.7768	0.5385
	*n*BR	0.9664	1.1522	0.6734	0.5881

with KG-like or graph-like datasets. This means not only rating information can be used to generate the recommendations, but we can integrate the other relations in the model to make the recommendations as well. For example, the FilmTrust dataset also provides *trust* property which is the relation between a pair of users. From trust information, we can model FilmTrust in the same way as in KG. Therefore, my proposed method can be applied to this dataset and then obtain the trust embedding vectors like rating embedding vectors, and can further use these trust embedding vectors to generate the recommendations. In other datasets, they may contain many explicit relations, e.g. *buy*, *view*, *click*, etc. which are relations between a pair of user and item. With these several kinds of relations, this proposed model can make better recommendations for users.

7.3.3 Discussion

In the previous section, I have shown that my proposed method can provide more accurate results in both rating prediction and ranking recommendation on all datasets.

Rater integration

Our proposed method is introduced to integrate *n* best rater in the model instead of only using the target user's information like CPE. I showed that including *n* best rater(s) can help the model to generate a better recommendation.

Although translation-based embedding models use every user's information in learning phrases, it does not utilize the benefit of *friendship*, which is the prominent point of CF, in the learning procedure. Therefore, I introduce to integrate *friendship*, in the form of *n* best rater(s), which is the prominent point of CF to the translation-based embedding model.

Table 7.10: Example of users rate items

Rating	User u_1	User u_2	User u_3
i_1	1	3	2
i_2	2	3	2
i_3	3	3	2
i_4	4	3	2
i_5	5	3	2

According to the experimental results, using the best rater's approximate rating vectors in the prediction step provides better results than a model without the best rater. However, using best rater's approximate rating vectors directly (e.g., without rating conversion method) and applying normalization provide lower accuracy than applying my proposed rating conversion before prediction.

Efficiency of Rating Conversion

Apart from n best rater integration, I also apply different rating conversion methods on those raters. The experimental result showed that my proposed method can provide a more accurate result than n best rater(s) without rating conversion, as well as applying normalization on n best rater(s). The details of each aspect are described as follows.

Rating Conversion vs. Actual Rating

For the efficiency of rating conversion, I will give some examples where the rating conversion is more effective than no rating conversion. For this purpose, I first give the example of users rate items in Table 7.10.

According to Table 7.10, suppose that user u_1 is a target user and item i_2 is a target item. From Table 7.10, the predicted rating should be '2'. Suppose that user u_3 is the target user's best rater, and these two users have different rating patterns. Suppose that the u_1's preference on i_2 (denoted by p_{u_1,i_2}) is '3.4', and u_3's preference on i_2 is '2.1'. Thus, the predicted rating of u_1 towards i_2 using Eq. 7.3.2 will be '2.75'. However, if we consider the different rating patterns of u_1 and u_3 and convert u_3's rating into u_1's perspective first, we will get a more accurate prediction. If the rating conversion method is applied to this case, suppose that u_3's preference on i_2 is converted to '1.8'. Therefore, the predicted rating of u_1 towards i_2 will be '2.6', which is closer to the actual rating than without rating conversion.

Rating Conversion vs. Normalization

This proposed rating conversion method also provides a better result than a normalization. I will give an example of why rating conversion is better than a basic normalization.

If two target users have different rating patterns but these two target users have the same set of rater(s), applying basic normalization is not enough. From Table 7.10, suppose that user u_1 and user u_2 are target users. If the target item is item i_2, normalized rating of u_1 on i_2 is '0.25' while normalized rating of u_2 on i_2 is '1'. Suppose that these two target users have the same rater which is user u_3, so the normalized rating of u_3 on i_2 is '1'. These two target users will have the same predicted rating '1' on i_2. Although the predicted rating is accurate on u_2, but it is a quite high score for user u_1 because his actual rating should be '0.25'. Therefore, the rating conversion method able to provide a more accurate prediction than a normalization under this circumstance.

Ranking Recommendation

Apart from the rating prediction, I also evaluate this proposed method by ranking recommendation. The experimental result showed that my proposed method on MovieLens (100K and 1M) dataset also provides better recommendation results in both terms of precision@10 and recall@10.

Moreover, the experimental results showed that only n best raters provide better results than CPE with n best raters. It can be concluded that the best rater(s) have an influence on the target user and the converted ratings of those best raters lead the model to generate good predictions.

7.3.4 Conclusion

In this work, I propose a translation-based embedding model with rating conversion in RSs by adjusting the obtained embedding vectors, which represent the relation between user and item.

Based on the experimental results and comparisons, I have demonstrated that my proposed method can provide more accurate predictions than the CPE for all datasets in terms of both rating prediction and ranking recommendation. Integrating friendship, which is n best rater's opinion, in the prediction step can provide a more accurate result than the original CPE method. Moreover, applying this proposed rating conversion method to the best rater's ratings provides better accuracy than both not applying the rating conversion method and the basic normalization method.

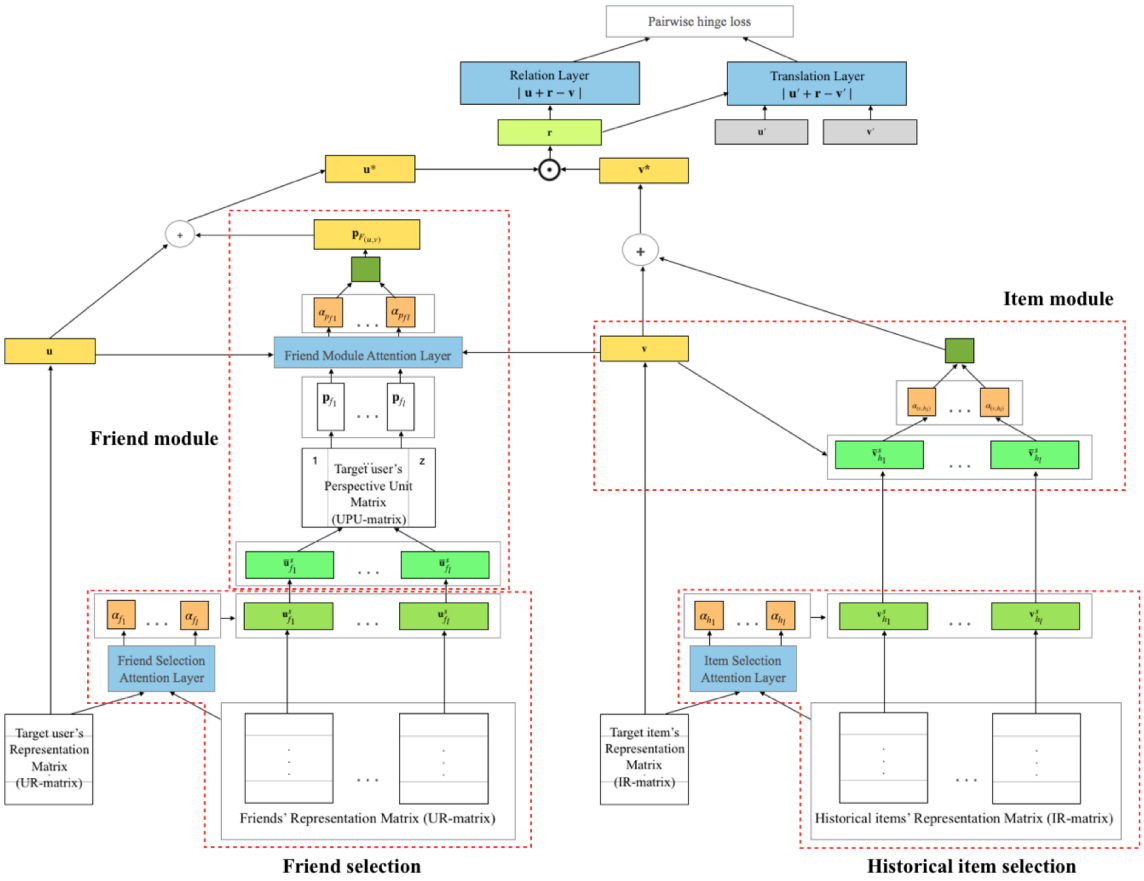

Figure 7.3: A workflow architecture of AHCF.

7.4 Attentive Hybrid Collaborative Filtering for Rating Conversion in Recommender Systems

In this subsection, an overview of the Attentive Hybrid Collaborative Filtering for Rating Conversion in Recommender Systems, i.e. *AHCF*, is presented. The workflow of AHCF is illustrated in Figure 7.3.

Features of the proposed model are as follows.

- Each of the users and items is converted to a dense matrix representation. Note that I introduce a dense matrix to represent a user and an item, unlike many RSs that use a single vector for this purpose. The details of this process are described in section 4.3.

- For each target user and target item pair, the model works on both sides parallelly as follows.

We select the set of friends (users who have rated the target item in the past) and the set of historical items (items that have been rated by the target user in the past). Note that in this work, I use a dataset that has no explicit friend relationship between users, so I defined the term 'friends' as users who have rated the target item in the past. This will be described in section 5.4.1.

Then I use the obtained set of friends as an input of the Friend Module. This step generates a summation of the weighted projected vectors of the target user's friends, effectively realizing both rating conversion and weight assignment. This will be described in section 5.4.2.

Similarly, I input the obtained historical items into the Item Module. This step generates a summation of item vectors of the target user's historical items based on the similarity between each historical item and the target item. This will be described in section 5.4.3.

- From the user side, the model aggregates the target user representation matrix into a single vector and combines it with the projected vectors of the target user's friends obtained in the previous step. From the item side, the model also aggregates the target items representation matrix into a single vector and combines it with the vectors of the target user's rated items obtained in the previous steps. It then calculates the latent relation vector $\mathbf{r}$, which is dependent on both user and item. This will be described in section 5.4.4.

- Finally, the model uses pairwise hinge loss and negative sampling to optimize $\|\mathbf{u}+\mathbf{r}-\mathbf{v}\| \approx 0$ which is the scoring function derived from TransE, one of KG embedding technique. This will be described in section 6.3.

To provide a better understanding of this model, the concrete examples are shown as follows. According to Figure 7.4, suppose that the target item is a cartoon movie. The model wants to predict how much the target user would like this target item. Our model will gather information from both the target user's friends (i.e., users who have rated the target item in the past) and the target user's historical items (i.e., items that the target user has rated in the past), to help the model predict the rating that target user would give to the target item. For the friend side, the model will compare similarities between the target user and each friend. Friends who are more similar to the target user would have more influence on the target user than friends who are less similar to the target user. In this case, suppose that user A (the girl with glasses), who rated 5 score to this movie, is more similar to the target user, then the target user tends to rate the target item with a high score. For the item side, the

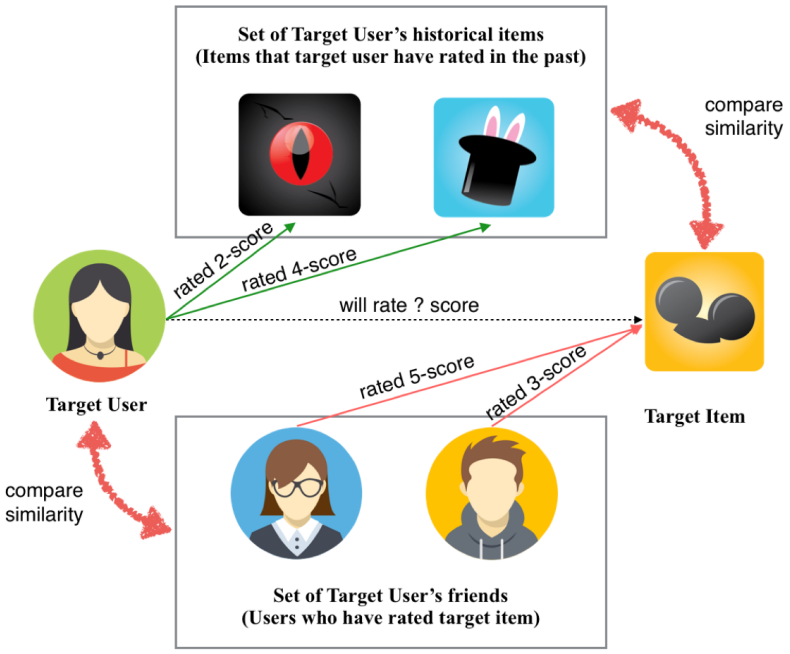

Figure 7.4: An illustration of a concrete example of my proposed model mechanism. Suppose there are target user and target item. The goal of RSs is to predict how much the target user will give to the target item. Our proposed method will gather the target user's friends and historical items to help model predict score of the target user toward the target item.

model will compare similarities between the target item and each historical item. Suppose that she rated 2 items in the past, which are horror movie and fantasy movie with 2 and 4 scores, respectively. My assumption is the target user tends to like similar items. In this case, suppose that the fantasy movie is more similar to the target item and she rated the fantasy movie with a 4-score, so she tends to rate the target item with a high score too. Note that this example will be used for the explanation entire this thesis.

7.4.1 Inference

After obtaining the optimized user's UR-matrices, user's UPU-matrices, and item's IR-matrices (i.e., $\mathbf{U}, \mathbf{V}, \mathbf{K}$), as well as all model parameters, the recommendation for each user is then reduced to a ranking problem among their negative items (i.e., items in the dataset that the target user has not rated in the past) based on estimated score $s(u, v)$ from Eq. 6.3.1.

7.4.2 Experimental Evaluation

In this subsection, I present details of the experiments. The experimental evaluation is designed to answer several research questions (**RQs**).

- **RQ1**: Is the proposed model more accurate than existing state-of-the-art methods for collaborative ranking?
- **RQ2**: Does the proposed UR-matrix and IR-matrix improve the model's performance?
- **RQ3**: Does the Friend Module and the Item Module affect the model's performance?
- **RQ4**: Does the combination of neighborhood-based CF and the model-based CF, namely hybrid CF, provide better results than only neighborhood-based CF or only model-based CF?

Datasets

I evaluated the method using three real datasets, which are FilmTrust[4] and two datasets from MovieLens[5]. FilmTrust is a recommendation dataset extracted from the entire FilmTrust website. MovieLens is a widely adopted benchmark dataset for CF in the application domain of recommending movies to users. In this work, I use two configurations of MovieLens, namely MovieLens100k and MovieLens1M. The numbers of users and items in each dataset are listed in Table 7.11.

[4]https://www.librec.net/datasets.html
[5]https://grouplens.org/datasets/movielens

Algorithm 1 Attentive Hybrid Collaborative Filtering* (AHCF*)

Input: User-item interaction matrix $\mathbf{R}$.
Output: Latent feature matrix $\mathbf{U}$, $\mathbf{V}$, $\mathbf{K}$ and model parameters Θ

Initialize $\mathbf{U}$, $\mathbf{V}$, $\mathbf{K}$, and models parameters Θ with normal distribution
repeat
  for all $(u,v) \in \mathbf{R}$ **do**
    Find a set of F_v and H_u
    for all $f \in F_v$ **do** ▷ Friend selection
      $\beta_f \leftarrow \mathbf{h}_u^T ReLU(\mathbf{W}\mathbf{u}_1\mathbf{U}_u + \mathbf{W}\mathbf{u}_2\mathbf{U}_f)$
      $\alpha_f \leftarrow \frac{exp(\beta_f)}{\sum_{j \in F_v} exp(\beta_j)}$
      $\overline{\mathbf{u}}_f^s \leftarrow \alpha_f \mathbf{u}_f^s$
    end for
    for all $h \in H_u$ **do** ▷ Historical item selection
      $\beta_h \leftarrow \mathbf{h}_i^T ReLU(\mathbf{W}\mathbf{i}_1\mathbf{V}_v + \mathbf{W}\mathbf{i}_2\mathbf{V}_h)$
      $\alpha_h \leftarrow \frac{exp(\beta_h)}{\sum_{j \in H_u} exp(\beta_j)}$
      $\overline{\mathbf{v}}_h^s = \alpha_h \mathbf{v}_h^s$
    end for
    for all $f \in F_v$ **do** ▷ Friend module
      $\mathbf{p}_f \leftarrow \mathbf{k}_u^s \odot \overline{\mathbf{u}}_f^s$
      $\beta_{p_f} \leftarrow \mathbf{h}^T ReLU(\mathbf{W}_1\mathbf{u} + \mathbf{W}_2\mathbf{p}_f + \mathbf{W}_3\mathbf{v})$
      $\alpha_{p_f} \leftarrow \frac{exp(\beta_{p_f})}{\sum_{j \in F_v} exp(\beta_{p_j})}$
    end for
    $\mathbf{p}_{F_{(u,v)}} \leftarrow \sum_{f \in F_v} \alpha_{p_f} \mathbf{p}_f$
    for all $h \in H_u$ **do** ▷ Item module
      $sim_{(v,h)} \leftarrow \mathbf{v}_v^T \overline{\mathbf{v}}_h^s$
      $\alpha_{(v,h)} \leftarrow Softmax(sim_{(v,h)})$
    end for
    $\mathbf{t}_{H_u} \leftarrow RELU(\sum_{h \in H_u} \alpha_{(v,h)} \overline{\mathbf{v}}_h^s)$
    $\mathbf{u}^* \leftarrow \lambda \mathbf{u} + (1-\lambda)\mathbf{p}_{F_{(u,v)}}$ ▷ Combined user
    $\mathbf{v}^* = \theta \mathbf{v} + (1-\theta)\mathbf{t}_{H_u}$ ▷ Combined item
    $\mathbf{r} \leftarrow \mathbf{u}^* \odot \mathbf{v}^*$ ▷ Generate latent relation
  end for
  for all $\vartheta \in \{\mathbf{U}, \mathbf{V}, \mathbf{K}, \Theta\}$ **do**
    update ϑ w.r.t $L = \sum_{(u,v) \in \Delta} \sum_{(u',v') \notin \Delta} max(0, s(u,v) + \Lambda - s(u',v'))$
  end for
until convergence
return $\mathbf{U}, \mathbf{V}, \mathbf{K}$ and Θ

Table 7.11: Datasets

Dataset	#Users	#Items	#Ratings
FilmTrust	1,508	2,071	35,497
MovieLens100K	943	1,682	100,000
MovieLens1M	6,040	3,706	1,000,209

Table 7.12: Evaluation results for experiments comparing with different methods

Dataset	Metrics	AHCF*	AHCF	AHCF-	LRML	NeuMF	GMF
FilmTrust	P@10	0.3124	0.3129	0.3090	0.3022	0.3121	**0.3196**
	Re@10	**0.6145**	0.6021	0.5968	0.5751	0.5580	0.5834
	F1	**0.4142**	0.4118	0.4072	0.3962	0.4003	0.4129
	MAP	**0.4511**	0.4446	0.4371	0.4293	0.4258	0.4403
	MRR	**0.5779**	0.5719	0.5622	0.5500	0.5322	0.5571
	NDCG	**0.6282**	0.6232	0.6158	0.6105	0.6020	0.6143
ML100K	P@10	**0.3458**	0.3177	0.2996	0.2858	0.2507	0.1856
	Re@10	**0.2337**	0.2095	0.1994	0.1907	0.1461	0.0883
	F1	**0.2789**	0.2524	0.2394	0.2288	0.1846	0.1200
	MAP	**0.2792**	0.2493	0.2417	0.2276	0.1810	0.1325
	MRR	**0.6371**	0.6047	0.5999	0.5658	0.4985	0.4342
	NDCG	**0.6133**	0.5878	0.5814	0.5684	0.5196	0.4639
ML1M	P@10	**0.3383**	0.3310	0.3129	0.2295	0.2016	0.1841
	Re@10	**0.1538**	0.1371	0.1357	0.0937	0.0763	0.0662
	F1	**0.2115**	0.1939	0.1893	0.1331	0.1107	0.0974
	MAP	**0.2286**	0.2151	0.2050	0.1376	0.1290	0.0997
	MRR	**0.6144**	0.5963	0.5821	0.4767	0.4413	0.3768
	NDCG	**0.5869**	0.5755	0.5643	0.5020	0.4825	0.4521

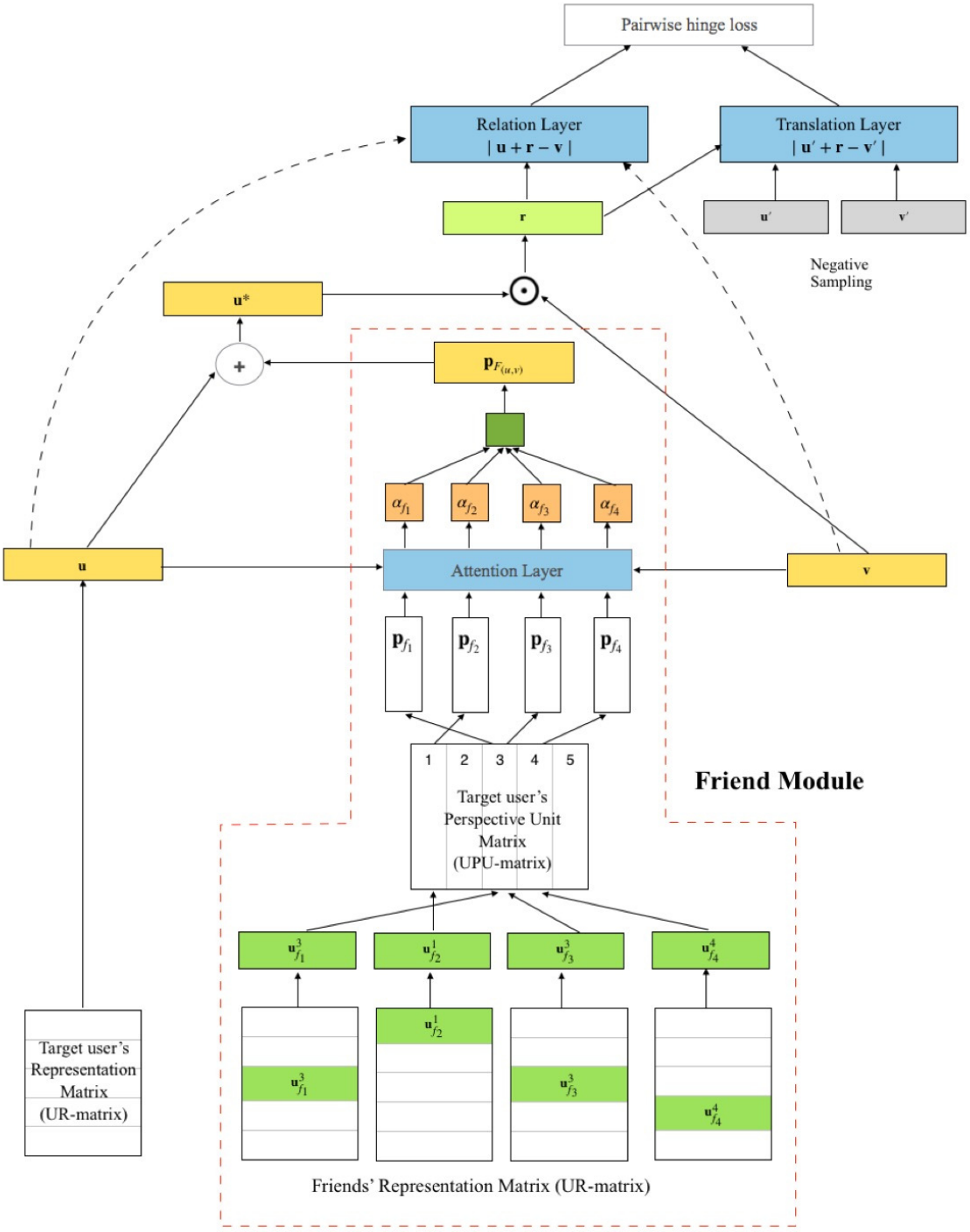

Figure 7.5: Schematic illustration of AHCF architecture as an end-to-end NN. This example assumes a user–item pair $\langle u, v \rangle$, with the user's four friends $\{f_1, f_2, f_3, f_4\}$ rating item v with rating scores $\{3, 1, 3, 4\}$, respectively.

Existing Baselines

For each of the three datasets, I conducted experiments where I compared my proposed model with the following existing methods.

- Latent relational matrix learning **LRML** [70]: an end-to-end attention-based memory-augmented neural architecture that models the relationship between users and items in metric space using latent relation vectors.
- Generalized matrix factorization **(GMF)** [37]: an implementation of MF [49], a standard baseline for CF that models the relationship between users and items using inner products.
- **NeuMF** [37]: a state-of-the-art unified framework combining MF with an MLP.
- **AHCF** [74]: my previous proposed method which contains only UR-matrix and friend module (See Figure 7.5).

Note that in the experiment, I call this extended model as **AHCF*** which is the improved version of **AHCF**, and **AHCF-** refers to the proposed model without the proposed friend module and the item module.

Implementation and Parameter Settings

In this work, I modified the source code of DeepRec, an open-source toolkit for deep-learning-based Recommendation systems [80], which includes several algorithms expressed in Python and Tensorflow[6]. DeepRec also includes the LRML, GMF, and NeuMF models.

All models were trained until convergence. I optimized the models by using AdaGrad [29], a stochastic optimization method that adapts the learning rate to the parameters. For all models, the dimension d was tuned amongst {50, 100, 200}, the learning rate was tuned amongst {0.1, 0.01, 0.001}, and the batch size is tuned amongst {256, 512, 1024}. For models that minimized the hinge loss, the margin γ was tuned amongst {0.1, 0.2, 0.5, 1.0}. For the proposed model, the attention size a was tuned amongst {50, 100, 200}, random number of friends and items are tuned among {5, 10, 20, 50}, λ and θ are tuned among {0.0, 0.5, 1.0}. For the LRML model, I set the number of memory slices to 20, as recommended in [70]. Note that I did not use pretrained models in the NeuMF model to enable a fair comparison. For simplicity, each training instance was paired with only a single negative sample. All embeddings and model parameters were normally initialized with a standard deviation of 0.01.

[6] https://www.tensorflow.org

Evaluation Protocol and Metrics

Each dataset was separated into two parts: 80% for training and the remaining 20% for testing. In the training phase, I sampled a single negative sampling for each user–item pair. After training the models, I performed the ranking recommendation task on the test set. Given a user–item pair $\langle u, v \rangle$, the negative samplings were all $\langle u, v' \rangle$, where v' is the set of items that had no interaction with the target user u. I used all the ranking-based metrics implemented in DeepRec [80], namely precision@n (P@10), recall@n (Re@10), F1 score, mean average precision (MAP), mean reciprocal rank (MRR), and normalized discounted cumulative gain (NDCG). These ranking-based metrics can evaluate the degree of effectiveness of the model.

Experimental Results

I compared the performance of the proposed model, AHCF*, with the existing methods listed above, including the previous version of AHCF [74]. Table 7.12 shows the empirical results for the proposed model and the existing methods for the three benchmark datasets. The results show that my proposed model was the best-performing model by most evaluation metrics and datasets. The proposed AHCF* can provide better results than the previous version AHCF [74]. This answers **RQ1** in the affirmative (i.e., the extended proposed model AHCF* can offer better effective recommendations in the collaborative ranking than existing systems).

For the FilmTrust dataset, my proposed model was the most accurate with respect to Re@10, MAP, MRR, and NDCG, but the GMF model performed best with respect to P@10. For the MovieLens100K and MovieLens1M datasets, my proposed model was the best-performing model for all evaluation metrics.

Adding Friend Module

In [74], I proposed the friend module and UR-matrix first. According to the experimental results in Table 7.12, the results have shown that AHCF (with the friend module) provides better results than AHCF- (without the friend module). It can be concluded that adding friend module to the model can help the model perform better and provide better results.

Adding Item Module

The previous version of AHCF [74] contains only the Friend Module for neighborhood-based CF and I have already shown that the Friend Module, which is neighborhood-based CF, can help the model-based network provide better results in [74]. The Friend Module can

be categorized as *user-based* neighborhood-based CF. In the extended version AHCF*, I add *item-based* neighborhood-based CF by proposing an additional Item Module.

To test the ability of the Item Module, I compare the performance of the current proposed model AHCF* both with and without the Item Module by setting $\lambda = 0.5$ for Eq. 5.4.13, and vary the value of θ in Eq. 5.4.14 which controls the usage of the Item Module with $[0.0, 1.0]$ with a step of 0.1. Figure 7.6 (A) shows P@10, Re@10, and NDCG on three datasets for this experimental setting. The experimental results have shown that $\theta = \{0.4, 0.5, 0.6\}$ provide better result than other θ for all metrics on every datasets.

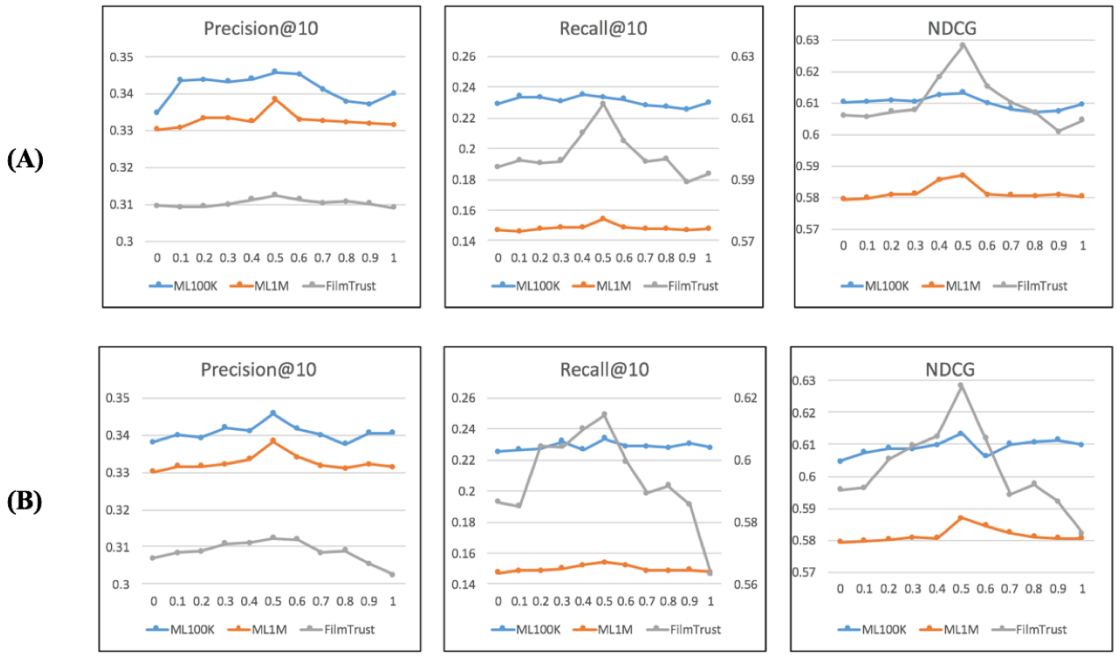

Figure 7.6: Performance of P@10, Re@10, and NDCG w.r.t. (A) different θ and (B) different λ on three datasets. Note that for Recall@10, because the results on FilmTrust are very different from ML100K and ML1M, so I use secondary axis to show results on FilmTrust

Furthermore, I also compare the performance of my proposed AHCF* both with and without the Friend Module by setting $\theta = 0.5$ for Eq. 5.4.14, and vary the value of λ in Eq. 5.4.13 which controls the usage of the Friend Module with $[0.0, 1.0]$ with a step of 0.1. Figure 7.6 (B) shows P@10, Re@10, and NDCG on three datasets for this experimental setting. The experimental results have shown that $\lambda = \{0.4, 0.5, 0.6\}$ provides better results than other λ for all metrics on every datasets.

Therefore, I select the results of $\lambda = 0.5$ and $\theta = 0.5$ to show the empirical results in Table 7.13, which shows the empirical results for this experimental setting on $\theta = \{0.0, 0.5, 1.0\}$.

Table 7.13: Evaluation results for experiments comparing the influence of the Item Module

Dataset	Metrics	θ		
		0.0	0.5	1.0
FilmTrust	P@10	0.3096	**0.3124**	0.3090
	Re@10	0.5939	**0.6145**	0.5919
	NDCG	0.6060	**0.6282**	0.6046
ML100K	P@10	0.3347	**0.3458**	0.3401
	Re@10	0.2290	**0.2337**	0.2298
	NDCG	0.6101	**0.6133**	0.6096
ML1M	P@10	0.3302	**0.3383**	0.3315
	Re@10	0.1473	**0.1538**	0.1483
	NDCG	0.5793	**0.5869**	0.5802

The results show that the model with $\lambda = 0.5$ and $\theta = 0.5$ provide better results for most metrics and datasets. It can be concluded that adding the item module to this proposed model can offer better accurate results.

Statistical hypothesis testing

Since the evaluation results of AHCF*, AHCF, and AHCF- are close to each other, I did the statistical test to show that evaluation results from the improved AHCF* are significantly different from AHCF and AHCF-. In this work, I use a paired samples t-test, a widely used statistical test that is used to compare the means of two samples when each observation in one sample can be paired with an observation in the other sample. In this case, I would like to test whether the predictions of the same set of users obtained from different methods are significantly different or not.

I use the statistics algorithm in Scipy library[7] to perform the paired samples t-test with the following hypotheses with a p-value threshold = 0.05:

- $H_0 : \mu_1 = \mu_2$
- $H_1 : \mu_1 \neq \mu_2$

The statistical t-test results are shown in Table 7.14. Since the obtained p-values from every datasets are less than the threshold of 0.05, then we can reject the null hypothesis of

[7]https://www.scipy.org

Table 7.14: Statistical hypothesis testing

Dataset	AHCF* vs AHCF	AHCF* vs AHCF-
FilmTrust	0.0105	0.0170
MovieLens100K	0.0445	0.0373
MovieLens1M	0.0246	0.0251

Table 7.15: Evaluation results for experiments comparing neighborhood-based model, model-based model, and hybrid model

Dataset	Metrics	Neighborhood	Hybrid	Model
FilmTrust	P@10	0.3083	**0.3124**	0.3087
	Re@10	0.5896	**0.6145**	0.5919
	NDCG	0.6007	**0.6282**	0.5974
ML100K	P@10	0.3396	**0.3458**	0.3373
	Re@10	0.2308	**0.2337**	0.2296
	NDCG	0.6086	**0.6133**	0.6092
ML1M	P@10	0.3304	**0.3383**	0.3320
	Re@10	0.1482	**0.1536**	0.1392
	NDCG	0.5794	**0.5869**	0.5805

equal averages and can interpret that the predictions from the improved version AHCF* are different from the previous version.

False Analysis

Since I perform the ranking recommendation task on the test set, which will generate top-k items for the target users. For evaluation, the ranking-based metrics is calculated based on the confusion matrix of recommendation results shown in Table 7.16.

The assumption of false positive case: why not relevant items are recommended to the target user?

- High quality target user's friends have rated those items in the past.
- Those items are similar to the target user's historical items.

The assumption of false negative case: why relevant items are not recommended to the target user?

- High quality target user's friends haven't rated those items in the past.
- Those items are not similar to the target user's historical items.

For this analysis, I choose some examples to explain. The first example is the example of user u_{100} from the ML100k dataset.

The generated top-10 recommendation list: [180, 171, 422, 215, 49, 227, 567, 81, 404, 221].

The set of relevant items that have not been recommended (**FN**): [20, 41, 53, 63, 76, 80, 87, 94, 117, 121, 131, 141, 150, 153, 156, 217, 218, 225, 226, 232, 239, 317, 322, 333, 339, 356, 424, 430, 442, 450, 455, 461, 469, 501, 558, 575, 606, 609, 657, 659, 664, 677, 685, 701, 718, 742, 771, 819, 1034].

The set of recommended items that are relevant (**TP**): [180, 171, 215, 49, 227, 567, 81, 404, 221].

The set of recommended items that are not relevant (**FP**): [422].

According to the example of this user, from the recommendation list, 9 items are True Positive (TP) and 1 item is False Positive (FP). Therefore, this recommendation is good. To investigate why items in the FN set are not recommended, I will find a set of high quality friends first, by selecting top-10 friends who have the highest similarity value to the target user. The ID of the top-10 friends of the target user are [824, 180, 356, 937, 140, 347, 707, 533, 871, 791].

Looking into the set of FN, there are 49 relevant items that have not been recommended. Among these 49 items, only 11 items have been rated by one of top 10 high quality friends. The rest of the 38 items have not been rated by the high-quality friends. I compare similarities between a set of FN and a set of u_{100}'s historical items. The average similarities between FN and u_{100}'s historical items is 0.001016. It means the items in FN set quite not similar to u_{100}'s historical items, so they are not recommended in the top-10 recommendation list to the target user.

For the False Positive case, there is only one item recommended to a target user. When looking into the set of top-10 friends of the target user. This recommended item was rated by the friends who have the highest similarity value to the target user (i.e., u_{824}). Therefore, this is one of the reasons why this item is recommended in the recommendation list.

The second example is user u_1 from the ML100k dataset.

The generated top-10 recommendation list is [285, 268, 293, 332, 301, 327, 244, 318, 747, 267].

Table 7.16: Confusion matrix of recommendation results

	Relevant	Not relevant
Recommended top-k	True Positive (TP)	False Positive (FP)
Not recommended	False Negative (FN)	True Negative (TN)

The set of relevant items that have not been recommended in the list (**FN**): [9, 12, 126, 254, 277, 278, 279, 307, 315].

The set of recommended items that are relevant (**TP**): [285, 268].

The set of recommended items that are not relevant (**FP**): [293, 332, 301, 327, 244, 318, 747, 267].

According to the example of user u_1, from the top-10 recommendation list, only 2 items are True Positive (TP) and 8 items are False Positive (FP). Compared to the above example of user u_{100}, this recommendation list is not quite good because only 2 relevant items are recommended.

To investigate why items in the FN set are not recommended, I do the same procedure as the above example. The ID of top-10 friends of user u_1 are [180, 732, 485, 893, 781, 459, 14, 78, 930, 935].

Then, I look into the set of FN, there are 9 items that have not been recommended in the recommendation list. First, I compare the similarities of items in FN with user u_1's historical items. The average similarity between FN and user u_1's historical items is 0.0488. Then, I compare the similarities of items in TP with user u_1's historical items. The average similarity between TP and user u_1's historical items is 0.1131.

From this investigation, it can be concluded that the items that have more similarity value to the target user's historical items will be recommended in the recommendation list, while the items that have less similarity value to the target user's historical items will not be recommended in the recommendation list.

For the false positive case, among the items in the FP set, I found that many users in the top-10 friends have rated those items in the FP set. Moreover, when I compare the similarities between items in the FP set and the user u_1's historical items, the average similarity value is 0.0531, which is higher than the FN set. Therefore, items in the FP set are recommended instead of items in the FN set because items in the FP set are more similar to the target user's historical items.

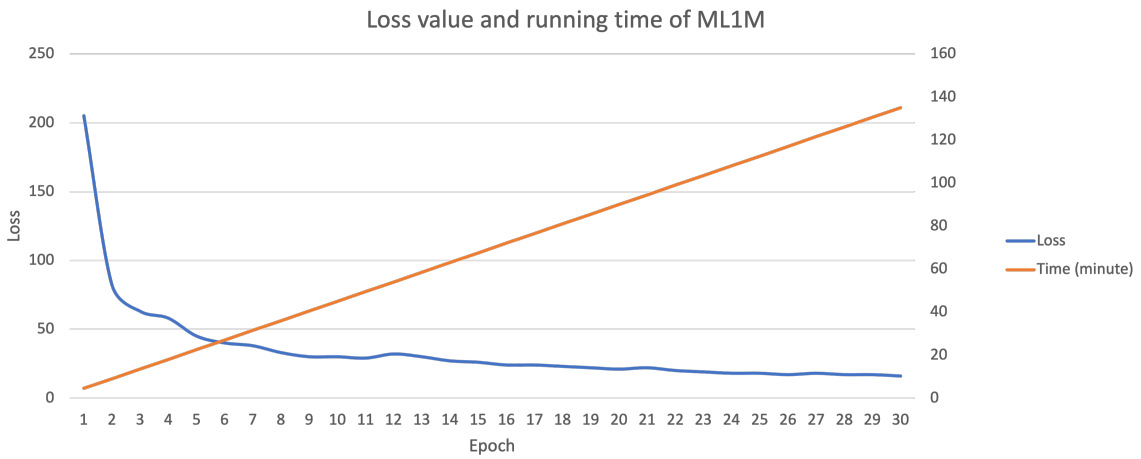

Figure 7.7: Loss value and running time of AHCF on ML1M

Computation and Time Complexity

According to the algorithm shown in Algorithm 1, this model's overall time complexity is $O(n^3)$. First, the model iterates through the entire user-item pairs in an outer loop. Then for each user-item pair, each module iterates through the data in an inner loop. This will be $O(n^2)$ for one epoch of training. For training, the model repeats its steps until convergence. Therefore, the overall complexity of training will be $O(n^3)$.

The experiments are conducted on GPU server Nvidia Tesla V100. In one epoch, I train the model with mini-batches. For ML1M, the training data is 800,000 records. If the batch size is 1024, it will be around 782 iterations per epoch. From the experiment, in the learning phase of ML1M, AHCF* takes around 0.35 seconds per iteration. Therefore, one epoch of training takes around 274 seconds or 4.57 minutes. Figure 7.7 shows the chart between loss and time used in each epoch.

By ablation analysis, which removes the neighbors selection module, friend module, and item module, the computation time is similar to the full version of AHCF* which is around 0.32 seconds per iteration. Therefore, adding these modules to the model does not increase time complexity significantly.

However, the larger dataset takes more time to train than the smaller dataset.

7.4.3 Discussion

In the previous subsection, I showed that my proposed model can provide the most accurate ranking recommendations for all datasets tested. In this subsection, I will discuss the evaluation results in several aspects as follows.

Performance of the Proposed UR-Matrix and IR-Matrix

In the previous work [74], I have proposed UR-matrix, the novel user representation for incorporating explicit feedback into the model via user presentation. Instead of representing the user with a single embedding vector, UR-matrix represents each user by matrix which stores rating information of users and expresses how each user rates items. The previous experimental result gives the comparative results that using the UR-matrix instead of using a traditional single vector can lead to more accurate ranking recommendations. These provide an answer to **RQ2** of UR-matrix, namely that using the proposed UR-matrix in the model can provide more accurate recommendation than the traditional embedding style.

In the extended model, I also propose an IR-matrix in addition to the UR-matrix which works as same as UR-matrix but for the item side. IR-matrix represents each item by matrix instead of a single embedding vector, which stores rating information of items and expresses how each item was rated by users. Table 7.13 gives the comparative results for the previous model [74] which contains only the UR-matrix and the current version which contains both the UR-matrix and IR-matrix. These provide an answer to **RQ2** of IR-matrix, namely that using the proposed IR-matrix in addition to the UR-matrix leads to more accurate ranking recommendations for all metrics and all datasets tested.

To conclude, this is because a dense matrix can express more meaningful user and item features than can a single embedding vector. In particular, a matrix in which each row is related to the user's rating and item's rating information can store more relevant and specific characteristics of the user and the item than can a single vector. Moreover, it is better suited and directly applicable to rating conversion, unlike the traditional approach to user embedding. In addition, integrating IR-matrix into the proposed model brings about richer information of the item side to the model.

Performance of the Friend Module and the Item Module

In previous work [74], I made three assumptions on user-based CF: 1) friends' opinions affect the target user's decision to choose items, 2) each user has their own rating patterns, and 3) friends can be more or less important to the target user. These assumptions are all implemented in the Friend Module.

In addition to the Friend Module, in the extended model, I also include item-based CF to the proposed model and made the assumption that target users shall prefer items that are similar to their historical interacted items. This assumption is implemented in the Item Module. The comparison between the previous model (i.e., without the Item Module) and

this new model (i.e., with the Item Module) has shown that the Item Module can help provide better recommendations.

I have shown three reasons why the Friend Module has a positive effect in the previous work. First, my UR-matrix integrates explicit feedback (ratings) into each friend representation matrix as input, thereby expressing how each user rates items and capturing this richer information in a way that a traditional single embedding vector cannot. Second, the UPU-matrix can convert each friend's specific ratings to match the target user's perspective in the Friend Module and thereby mitigate the improper rating-range problem. Finally, a nonuniform weight is assigned to each friend to indicate the relative importance of each friend to a pair of target user–item.

The reasons why the Item Module can help the model provide better results are as follows. First, similar to the Friend Module, the proposed IR-matrix integrates explicit feedback into each historical item representation matrix as input, so it can express how each item is rated by users and also capture more information than a single vector. Second, according to my assumption that users shall prefer items that are similar to their historical interacted items, this module can transfer the target user's historical items into the proposed model based on the similarity between the target user's historical items and the target item.

These provide an answer to **RQ3**, namely that the Friend Module and the Item Module enhance the model's performance.

Combination of neighborhood-based CF and model-based CF

Since my assumption is combining neighborhood-based CF with model-based CF, namely hybrid CF, can provide better recommendation results than only neighborhood-based CF alone or only model-based CF alone. To answer **RQ4**, I tune the hyper-parameters λ and θ which control the weight between neighborhood-based CF and model-based CF. In Table 7.15, the comparison has shown that hybrid CF, in which combining neighborhood-based CF and model-based CF (i.e., $\lambda = 0.5$ and $\theta = 0.5$) can help provide better recommendations than neighborhood-based CF alone (i.e., $\lambda = 0.0$ and $\theta = 0.0$) or model-based CF alone (i.e., $\lambda = 1.0$ and $\theta = 1.0$).

Because neighborhood-based CF and model-based CF are both functional in RSs and they both provide their own advantages. Thus, combining neighborhood-based CF and model-based CF in equivalent or similar ratios in RSs can provide better recommendation results than only neighborhood-based CF or only model-based CF. This can provide an answer to **RQ4**.

Performance comparison between AHCF* and AHCF

According to Table 7.12, the result shows that the improved version AHCF* provides better results than the previous version AHCF [74]. However, I notice that the improvement of the results over AHCF is less than 2% for the FilmTrust dataset, whereas it is less than 10-12% in ML100K and ML1M datasets.

One possible factor is rating ranges s of each dataset are not equal. In this paper, FilmTrust's rating range is a set of $\{1,2,3,4\}$, while rating range of ML100K and ML1M are $\{1,2,3,4,5\}$. Since this work focuses on and pays attention to the individual rating score, the proposed AHCF* may work well on the dataset that provides a wider range of ratings. In the previous work [74], I have shown that the model representing the user in the s-rows matrix achieves better results than the model in a single embedding vector.

I evaluate this assumption by a small experiment on ML100K and ML1M datasets by separating rating range s-score into binary score $[1,2]$. I set the threshold T and binary score $B(s)$ are defined as follows.

$$B(s) = \begin{cases} 1, s < T \\ 2, s \geq T \end{cases}$$

The results obtained from AHCF* on binary rating representation are better than a single embedding vector but less than the original s-score. Moreover, my framework contains a rating conversion method which converts specific ratings between a specific pair of user–item. The conversion method should work better on datasets that contain a wider rating range. Therefore, we can interpret that the number of rating range affects AHCF* performance.

Another possible factor that affects the performance of the model is the sparsity of data. I notice that the density of the FilmTrust dataset is 1.14%, whereas the density of ML100K and ML1M are 6.30% and 4.47%, respectively. It might be concluded that the sparsity of data affects the model performance. However, we have to carefully investigate this issue in several aspects. There might be another factor that affects the performance of the proposed AHCF*.

Performance of AHCF* on different number of ratings distributions

I also study how the distribution of ratings affects the AHCF*. Our assumption is that different distributions of ratings affect the model. If the dataset contains a number of high scores more than a number of less scores, the model should perform better. I generate synthetic data with different numbers of rating distributions as follows.

1. Users always rate items with a low score more than a high score

Table 7.17: Evaluation results for experiments on the synthetic dataset with different number of rating distribution

Metrics	1	2	3
P@10	0.2285	0.4025	**0.7575**
Re@10	0.0893	0.1272	**0.2017**
NDCG	0.6372	0.7492	**0.9137**

2. Users always rate items with a low score and high score equally

3. Users always rate items with a high score more than a low score

I generate a full interaction matrix with rating ranges of {1,2,3,4,5} between 200 users and 200 items with three different numbers of ratings above. The number of interactions is 40,000 records for all three datasets. I do the experiment on these three synthetic datasets with AHCF* by the setting of $\lambda = 0.5$ and $\theta = 0.5$.

The results in Table 7.17 have shown that AHCF* performs best results on the 3rd synthetic data with users always rate items with a high score more than a low score and performs worst results on the 1st synthetic data with users always rate items with a low score more than a high score.

Because AHCF* evaluates the result by ranking recommendation, the model will recommend items with higher predicted scores. If the dataset contains a high number of high rating scores {4,5}, AHCF* learns user and item representations from those given high rating scores. Therefore, we can expect to obtain the user and item representations which correspond to those given high scores and the recommendation should be good as well. In contrast, if the dataset contains a high number of low rating scores {1,2}, AHCF* learns user and item representations from those that give low rating scores. Then the obtained user and item representations will be corresponded to low rating scores which leads to bad recommendations.

Moreover, I have learned from this experiment that explicit feedback (i.e., rating score) is important for a recommendation. Even though these 3 sets of synthetic data contain the same number of records, the results are different because ‘user rates item’ does not interpret that ‘user likes item’.

7.4.4 Conclusion

In this work, I propose an end-to-end KG-based attentive hybrid CF NN architecture for rating conversion in RSs and ranking recommendations, using explicit feedback. Different from traditional RSs, I propose efficient and novel input representations, which are UR-matrix and IR-matrix, to enable the direct incorporation of explicit feedback (ratings) into the model via input representation. The UR-matrix and IR-matrix can store more relevant information and specific characteristics of users and items, respectively. Our proposed model includes: (1) a Friend Module (user-based CF) that first converts friends' ratings to match the target user's perspective, and then assigns a nonuniform individual weight to each user, (2) an Item Module (item-based CF) that can capture relations between the target item and target user's historical items based on their similarity. The experimental results show that the proposed model can provide more accurate results than existing methods.

7.5 Discussion

In subsection 7.2, 7.3 and 7.4, the details of TrustAnt [72], TransRS [73] and AHCF [74] are presented respectively. Since the main purpose of this thesis is focusing on a hybrid CF RSs, therefore, in this subsection, I will discuss the differences between these two proposed models (i.e., TransRS and AHCF) because TrustAnt is a neighborhood-based CF model.

Even though these TransRS and AHCF are categorized as the hybrid CF model in RSs, they do not share the same main idea. Also, they perform and generate the recommendations in different ways. The brief comparison is described in Table 7.18.

7.5.1 Model Architecture

The former one TransRS [73] is a translation-based embedding model which consists of three main steps. First, the model feeds positive triplets of $(user, item, rating)$ in TransE [12] KG embedding model. By applying constraints like those in CPE [77], in the first step, the user, item, and relation embedding vectors are generated, which will be used in the next step. Second, for each pair of target user–item, find a set of n target user's best friends and then perform the PCA-based rating conversion method which is described in 4.2. After the second step, the converted embedding vectors of n best friends are obtained. Finally, these obtained embedding vectors will be fed in the rating prediction step and predict the score that the target user will give to the target item. From these steps, TransRS [73] can be categorized as a building blocks model in which each step is a module that takes an input from the previous one and generates output for the next. This kind of architecture may look

Table 7.18: Comparison between TransRS [73] and AHCF [74]

Methods	TransRS	AHCF
Model architecture	building blocks	end-to-end NN
Model parameters	only model-based block contains model parameters	one set model parameters
Rating prediction and output	scalar form	vector form
Optimization and learning	only in model-based blocks	in whole end-to-end model
Evaluation	rating prediction & ranking recommendation	ranking recommendation

quite simple to understand and implement when comparing the end-to-end models since each module can handle its own objective without relying on the others. However, it requires a lot of computational resources. Moreover, the building blocks model is outdated, since researchers are recently more interested in an end-to-end, which are more practical and easier to reproduce the results.

While the latter one, AHCF [74] is categorized as an end-to-end NN model, which is a modern approach and more practical than the building blocks model. Unlike the former model, even though this end-to-end model consists of several modules, it can train the given train dataset and can generate the recommendation within the model itself in one single optimization. Because of its effectiveness in utilization and reproducing the results, recent research in the RSs field usually proposed recommendation models based on this end-to-end model.

7.5.2 Model parameters

Model parameters are properties of training data that will be learned during the learning process in the model-based CF model. Since TransRS [73] is a model that consists of building blocks, only model-based blocks contain the model parameters, which will be optimized within their own blocks. In TransRS [73], the first step of generating user, item, and relation embeddings is implemented by TransE embedding techniques, all embeddings are model parameters and they are trained and optimized within the first step.

On the other hand, based on the end-to-end model properties, the model parameters of AHCF [74] are trained and optimized in a single model, so it is like a one-stop service which is better than the building blocks.

7.5.3 Rating prediction and output

In RSs, rating prediction is one of the most important steps. Generally, RSs will predict the rating score of the target user towards the target item, and those predicted rating scores are usually in a scalar form.

According to my proposed models, the rating prediction of each model is different. In TransRS [73], because it is a hybrid CF model which is implemented based on 'applying model-based CF idea to neighborhood-based CF model', the prediction step is similar to the traditional neighborhood-based CF. It can predict the rating score of the target user towards to the target item in scalar form. The output of this model is a predicted rating score in scalar value.

Unlike TransRS [73], AHCF [74] does not predict the rating score of the target user towards the target item in scalar form. Based on the model architecture and its optimization, the objective of the model is to predict relative distances between inputs. Therefore, the task of the model is not a rating prediction, but rather a ranking recommendation. Therefore, this model does not predict the concrete rating score in scalar form like traditional neighborhood-based CF models do. It generates a specific latent relation vector of the target user towards the target item.

7.5.4 Optimization and learning

Because the prediction step of TransRS [73] is a neighborhood-based CF model which is based on the memory-based approach, it does not contain any optimization and learning processes. On the other hand, AHCF [74] is an end-to-end NN CF model which essentially requires the optimization and learning processes as important parts of the model. In this work, I use pairwise hinge loss and AdaGrad [29] as the optimization.

Unlike other loss functions, such as cross-entropy loss or MAE loss, whose objective is to learn to predict directly a label, a value, or a set or values given an input, the objective of the pairwise hinge loss is to predict relative distances between inputs. AHCF tries to model the users and items embeddings based on the idea of TransE. Therefore, this pairwise hinge loss is suitable for the model.

7.5.5 Evaluation

In RSs, to evaluate the performance of the model, there are two popular approaches, which are rating prediction and item ranking recommendation. In TransRS [73], since the output of

the model is in a scalar predicted rating form, it is possible to do both rating prediction and item ranking recommendation.

On the other hand, in AHCF [74], only the item ranking recommendation is implemented since this model is implemented and optimized by a pairwise hinge loss, which is the ranking loss, and the output is in a vector form rather than a scalar rating. Therefore, AHCF cannot do the rating prediction to evaluate the performance of the model.

Nowadays, item ranking recommendation becomes more popular than rating prediction because it is more practical than rating prediction. According to the scenario in the real world, for example, when a user wants to watch a movie, the RSs always gives the user a list of movies he/she might like. This is the item ranking recommendation which is generated based on the preference of each user. On the other hand, rating prediction usually generates prediction only of a specific pair of user–item in the test dataset. Therefore, item ranking recommendation is more practical than rating prediction.

Chapter 8

Summary

8.1 Summary

This thesis presents a novel approach for the hybrid CF RSs which can leverage and combine the advantages of both neighborhood-based CF approach and model-based CF approach in building the efficient hybrid CF RSs. As a hybrid CF model, the main challenge is how to combine neighborhood-based approach and model-based approach together. Moreover, each approach still suffers from their own problems, so these problems are need to be solved during proposing a novel hybrid CF RSs.

The main research which contributes to this thesis is Attentive Hybrid Collaborative Filtering for Rating Conversion in Recommender Systems (AHCF) [74].

First, an approach to utilize neighbors into a hybrid CF model is proposed in [74]. Since the main idea of this thesis is to apply the neighborhood-based idea into the model-based CF model, so the model structure, including user and item representations are newly proposed in this work. Unlike traditional existing CF RSs models, the novel user and item representations, called user representation matrix (UR-matrix) and item representation matrix (IR-matrix) are introduced to this model. In order to store neighbors' information which is usually in the form of explicit feedback, UR-matrix and IR-matrix can store and specify characteristics of each user and item respectively.

Second, in the neighborhood-based CF approach, the issues to consider in this thesis are: 1) different influence of neighbors on the target user and 2) different rating patterns of users. According to the first issue, all friends' influences are not equal from the target user's perspective. The closest friend should have the most influence on the target user and should participate the most role in the model. Likewise, for the item side, the historical item that is most similar to the target item should have more influence than the less similar historical item. Thus, the friends and historical items selection module are proposed to solve this issue.

In addition to the first issue, the second issue of the neighborhood-based CF approach, which is different rating patterns of users, is solved in the model too. Due to the improper rating-range problem that occurs when users have different rating patterns, the rating of friends should be converted into the target user's perspective before prediction. The rating conversion between a pair of user is proposed in the friend module which can be categorized as a user-based CF that first converts friends' ratings to match the target user's perspective, and then assigns a nonuniform individual weight to each user Moreover, the item module which can be categorized as item-based CF is proposed to capture relations between the target item and target user's historical items based on their similarity.

Third, in the model-based CF approach, we need to consider how to model the relationship between user and item. Some of the model-based CF models try to construct user–item relationship by minimizing the distance between user and item in vector space. In order to prevent the geometric inflexibility problem, the TransE [12] concept, one of the most popular translation-based KG embedding models, is introduced to this proposed model. In this work, the latent relation between a pair target user–item is constructed from two components: 1) combination between the target user and his/her friends (obtained from Eq. 5.4.13) and 2) combination between target item and target user's historical items (obtained from Eq. 5.4.14). Then the optimization of this model is learned based on the TransE idea.

With AHCF model that consists of all proposed components to solve the aforementioned problems in both neighborhood-based and model-based CF approaches, the experimental results have shown that the proposed model provides better effectiveness than existing methods.

In addition to the main work of AHCF, I have proposed two more models, which are Translation-based Embedding model for Rating Conversion in RSs (TransRS) [73] and Integrating the importance levels of friends into trust-based ant-colony RSs (TrustAnt) [72]. The proposed model in [73] is a hybrid CF model which tries to combine neighborhood-based and model-based CF approach into the model. Although TransRS is a hybrid CF RS, the model architecture is a pipeline model instead of the end-to-end model.

The proposed model TrustAnt in [72] is a neighborhood-based CF model. This model is included in this thesis because it consists of a rating conversion method to solve the improper rating-range problem, which is one of the main challenges of this thesis.

8.2 Future Plan

For future work, there are three further challenges that are possible to be extended.

First, there is room to improve the rating conversion method. Inspired by Coherence and inconsistencies in rating behavior: estimating the magic barrier of RSs [64], they observed that user coherence affects the recommendation results. They separated users into 2 groups, which are 1) easy to predict (more coherence) and 2) difficult to predict (less coherence), and they show that training and testing the model with the easy group can provide more accurate results than training and testing with the difficult group. In their model, they extract the coherence of users by genre of movies. By integrating the coherence of users, it might increase the effectiveness of the model.

Second, neighbors selection method can be improved by additional information. Since in [72], I use trust information to form the graph of users and use the ACO concept with trust value to find high-quality neighbors. Because this model is the neighborhood-based CF model and it might not practical for recent problems, so in future, I plan to include the additional information (e.g., trust relation) to the end-to-end NN hybrid CF model to find high-quality neighbors and hope that the model can learn better user and item representations.

Finally, I plan to generate explainable recommendations. According to the [83], explainable recommendation refers to personalized recommendation algorithms that address the problem of **why** - they not only provide users or system designers with recommendation results, but also explanations to clarify why such items are recommended. Explainable AI aims to develop models that can explain their (or other model's) decisions for system designers or normal users. There are several styles of recommendation explanations, such as explanations based on relevant users or items, which are closely related to the idea behind user-based or item-based CF methods. According to the architecture of the AHCF model, which consists of both user-based and item-based CF concepts, it is possible to answer the problem of why such items are recommended in more detail. However, it need more experimental evaluation, so I would like to take this idea for the next step of future work.

Publication list

Journals

- P. Tengkiattrakul, S. Maneeroj, and A. Takasu, “Integrating the Importance Levels of Friends into Trust-based Ant-colony Recommender Systems”, International Journal of Web Information System, Vol. 15 No. 1, pp. 28-46, 2019.

International conference

- P. Tengkiattrakul, S. Maneeroj, and A. Takasu, “Translation-based Embedding Model for Rating Conversion in Recommender Systems”, In IEEE/WIC/ACM International Conference on Web Intelligence (WI ’19), pp. 217–224, October 2019. (Full Paper)
- P. Tengkiattrakul, S. Maneeroj, and A. Takasu, “Attentive Hybrid Collaborative Filtering for Rating Conversion in Recommender Systems”, In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds) Web Engineering (ICWE 2021), Springer, Cham, pp.151–165, May 2021. (Full Paper)

Bibliography

[1] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.

[2] Aggarwal, C. C., Wolf, J. L., Wu, K.-L., and Yu, P. S. (1999). Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 201–212, New York, NY, USA. ACM.

[3] Ahamed, M. T. and Afroge, S. (2019). A recommender system based on deep neural network and matrix factorization for collaborative filtering. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–5.

[4] Ayub, M., Ghazanfar, M. A., Mehmood, Z., Saba, T., Alharbey, R., Munshi, A. M., and Alrige, M. A. (2019). Modeling user rating preference behavior to improve the performance of the collaborative filtering based recommender systems. *PLOS ONE*, 14(8):1–29.

[5] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.

[6] Bedi, P. and Sharma, R. (2012a). Trust based recommender system using ant colony for trust computation. *Expert Systems with Applications*, 39(1):1183–1190.

[7] Bedi, P. and Sharma, R. (2012b). Trust based recommender system using ant colony for trust computation. *Expert Systems with Applications*, 39(1):1183–1190.

[8] Beel, J., Gipp, B., Langer, S., and Breitinger, C. (2016). Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338.

[9] Bellaachia, A. and Alathel, D. (2016a). Improving the recommendation accuracy for cold start users in trust-based recommender systems. *International Journal of Computer and Communication Engineering*, 5:206–214.

[10] Bellaachia, A. and Alathel, D. (2016b). Improving the recommendation accuracy for cold start users in trust-based recommender systems. *International Journal of Computer and Communication Engineering*, 5:206–214.

[11] Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2014). A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259.

[12] Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2787–2795, USA. Curran Associates Inc.

[13] Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

[14] Catherine, R. and Cohen, W. (2017). Transnets: Learning to transform for recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 288–296, New York, NY, USA. Association for Computing Machinery.

[15] Chalermpornpong, W., Maneeroj, S., and Atsuhiro, T. (2013). Rating pattern formation for better recommendation. In *2013 24th International Workshop on Database and Expert Systems Applications*, pages 146–151.

[16] Chen, C., Zhang, M., Liu, Y., and Ma, S. (2018a). Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1583–1592, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[17] Chen, C., Zhang, M., Liu, Y., and Ma, S. (2019a). Social attentional memory network: Modeling aspect- and friend-level differences in recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, pages 177–185, New York, NY, USA. Association for Computing Machinery.

[18] Chen, J., Zhang, H., He, X., Nie, L., Liu, W., and Chua, T.-S. (2017). Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 335–344, New York, NY, USA. Association for Computing Machinery.

[19] Chen, R., Hua, Q., Chang, Y.-S., Wang, B., Zhang, L., and Kong, X. (2018b). A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. *IEEE Access*, 6:64301–64320.

[20] Chen, T. and Guestrin, C. (2016). XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

[21] Chen, W., Cai, F., Chen, H., and Rijke, M. D. (2019b). Joint neural collaborative filtering for recommender systems. *ACM Trans. Inf. Syst.*, 37(4).

[22] Chen, Y., Wu, C., Xie, M., and Guo, X. (2011). Solving the sparsity problem in recommender systems using association retrieval. *Journal of Computers*, 6:1896–1902.

[23] Cunningham, P. and Delany, S. J. (2022). k-nearest neighbour classifiers - a tutorial. *ACM Computing Surveys*, 54(6):1–25.

[24] Deng, Z., Huang, L., Wang, C., Lai, J., and Yu, P. S. (2019). Deepcf: A unified framework of representation learning and matching function learning in recommender system. *CoRR*, abs/1901.04704.

[25] Desrosiers, C. and Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 107–144. Springer.

[26] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 601–610, New York, NY, USA. Association for Computing Machinery.

[27] Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., and Zhang, F. (2017). A hybrid collaborative filtering model with deep structure for recommender systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 1309–1315. AAAI Press.

[28] Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

[29] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

[30] Dziugaite, G. K. and Roy, D. M. (2015). Neural network matrix factorization. *CoRR*, abs/1511.06443.

[31] Gong, S., Ye, H., and Dai, Y. (2009). Combining singular value decomposition and item-based recommender in collaborative filtering. In *2009 Second International Workshop on Knowledge Discovery and Data Mining*, pages 769–772.

[32] Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 439–446, USA. American Association for Artificial Intelligence.

[33] Grčar, M., Mladenić, D., Fortuna, B., and Grobelnik, M. (2006). Data sparsity issues in the collaborative filtering framework. In *Advances in Web Mining and Web Usage Analysis: 7th International Workshop on Knowledge Discovery on the Web, WebKDD 2005*, pages 58–76.

[34] Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., and He, Q. (2020). A survey on knowledge graph-based recommender systems. *ArXiv*, abs/2003.00911.

[35] Han, X., Cao, S., Lv, X., Lin, Y., Liu, Z., Sun, M., and Li, J. (2018). Openke: An open toolkit for knowledge embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 139–144. Association for Computational Linguistics.

[36] Hangal, S., MacLean, D., Lam, M. S., and Heer, J. (2010). All friends are not equal: Using weights in social graphs to improve search. In *Workshop on Social Network Mining & Analysis, ACM KDD*.

[37] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 173–182, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[38] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 230–237, New York, NY, USA. Association for Computing Machinery.

[39] Hsieh, C.-K., Yang, L., Cui, Y., Lin, T.-Y., Belongie, S., and Estrin, D. (2017). Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 193–201, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[40] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 263–272, USA. IEEE Computer Society.

[41] Huang, Z., Chung, W., Ong, T.-H., and Chen, H. (2002). A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '02, pages 65–73, New York, NY, USA. Association for Computing Machinery.

[42] Ji, G., He, S., Xu, L., Liu, K., and Zhao, J. (2015). Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, Beijing, China. Association for Computational Linguistics.

[43] Jin, R. and Si, L. (2004). A study of methods for normalizing user ratings in collaborative filtering. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 568–569, New York, NY, USA. ACM.

[44] Jin, R., Si, L., Zhai, C., and Callan, J. (2003). Collaborative filtering with decoupled models for preferences and ratings. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 309–316, New York, NY, USA. ACM.

[45] Juan, W., Yue-xin, L., and Chun-ying, W. (2019). Survey of recommendation based on collaborative filtering. *Journal of Physics: Conference Series*, 1314(1):012078.

[46] Kaleroun, A. and Batra, S. (2014a). Collaborating trust and item-prediction with ant colony for recommendation. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 334–339, Los Alamitos, CA, USA. IEEE Computer Society.

[47] Kaleroun, A. and Batra, S. (2014b). Collaborating trust and item-prediction with ant colony for recommendation. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 334–339.

[48] Klema, V. and Laub, A. (1980). The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176.

[49] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[50] Lathia, N., Hailes, S., and Capra, L. (2008). Trust-based collaborative filtering. In Karabulut, Y., Mitchell, J., Herrmann, P., and Jensen, C. D., editors, *Trust Management II*, pages 119–134, Boston, MA. Springer US.

[51] Lee, D. D. and Seung, H. S. (2000). Algorithms for non-negative matrix factorization. In *Proceedings of the 13th International Conference on Neural Information Processing Systems - NIPS '00*, pages 535–541.

[52] Li, M., Tei, K., and Fukazawa, Y. (2019). An efficient co-attention neural network for social recommendation. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, pages 34–42, New York, NY, USA. Association for Computing Machinery.

[53] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2181–2187. AAAI Press.

[54] Liu, H., Hu, Z., Mian, A., Tian, H., and Zhu, X. (2014). A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems*, 56(C):156–166.

[55] Liu, Y., Wang, S., Khan, M. S. A., and He, J. (2018). A novel deep hybrid recommender system based on auto-encoder with neural collaborative filtering. *Big Data Min. Anal.*, 1:211–221.

[56] Massa, P. and Avesani, P. (2007). Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, page 17–24, New York, NY, USA. Association for Computing Machinery.

[57] Mishra, N., Chaturvedi, S., Vij, A., and Tripathi, S. (2021). Research problems in recommender systems. *Journal of Physics: Conference Series*, 1717(1):012002.

[58] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA. Omnipress.

[59] Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., and Ferro, E. (2018). Translational models for item recommendation. In Gangemi, A., Gentile, A. L., Nuzzolese, A. G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J. Z., and Alam, M., editors, *The Semantic Web: ESWC 2018 Satellite Events*, pages 478–490, Cham. Springer International Publishing.

[60] Pirasteh, P., Bouguelia, M.-R., and Santosh, K. C. (2021). Personalized recommendation: an enhanced hybrid collaborative filtering. *Advances in Computational Intelligence*, 1(4):1.

[61] Qiao, C., Huang, B., Niu, G., Li, D., Dong, D., He, W., Yu, D., and Wu, H. (2018). A new method of region embedding for text classification. In *International Conference on Learning Representations*.

[62] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2010). *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition.

[63] Ricci, F., Shapira, B., and Rokach, L. (2015). *Recommender systems handbook, Second edition*. Springer New York, NY.

[64] Said, A. and Bellogín, A. (2018). Coherence and inconsistencies in rating behavior: Estimating the magic barrier of recommender systems. *User Modeling and User-Adapted Interaction*, 28(2):97–125.

[65] Salakhutdinov, R. and Mnih, A. (2007). Probabilistic matrix factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems - NIPS '07*, pages 1257–1264.

[66] Sharma, M., Mittal, R., Bharati, A., Saxena, D., and Singh, A. (2022). A survey and classification on recommendation systems.

[67] Shlens, J. (2014). A tutorial on principal component analysis. *CoRR*, abs/1404.1100.

[68] Sitkrongwong, P., Maneeroj, S., and Takasu, A. (2019). Multi-criteria rating conversion without relation loss for recommender systems. *International Journal of Computers and Applications*.

[69] Socher, R., Chen, D., Manning, C. D., and Ng, A. Y. (2013). Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, page 926–934, Red Hook, NY, USA. Curran Associates Inc.

[70] Tay, Y., Anh Tuan, L., and Hui, S. C. (2018). Latent relational metric learning via memory-based attention for collaborative ranking. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 729–739, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[71] Tengkiattrakul, P., Maneeroj, S., and Takasu, A. (2016). Applying ant-colony concepts to trust-based recommender systems. In *Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services*, iiWAS '16, page 34–41, New York, NY, USA. Association for Computing Machinery.

[72] Tengkiattrakul, P., Maneeroj, S., and Takasu, A. (2019a). Integrating the importance levels of friends into trust-based ant-colony recommender systems. *International Journal of Web Information Systems*, 15(1):28–46.

[73] Tengkiattrakul, P., Maneeroj, S., and Takasu, A. (2019b). Translation-based embedding model for rating conversion in recommender systems. In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI '19, pages 217–224, New York, NY, USA. Association for Computing Machinery.

[74] Tengkiattrakul, P., Maneeroj, S., and Takasu, A. (2021). Attentive hybrid collaborative filtering for rating conversion in recommender systems. In Brambilla, M., Chbeir, R., Frasincar, F., and Manolescu, I., editors, *Web Engineering*, pages 151–165, Cham. Springer International Publishing.

[75] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[76] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

[77] Wang, X., Xu, C., Guo, Y., and Qian, H. (2016). Constrained preference embedding for item recommendation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2139–2145. AAAI Press.

[78] Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1112–1119. AAAI Press.

[79] Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases.

[80] Zhang, S., Tay, Y., Yao, L., Wu, B., and Sun, A. (2019a). Deeprec: An open-source toolkit for deep learning based recommendation. *CoRR*, abs/1905.10536.

[81] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019b). Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1).

[82] Zhang, Y., Ai, Q., Chen, X., and Wang, P. (2018). Learning over knowledge-base embeddings for recommendation. *CoRR*, abs/1803.06540.

[83] Zhang, Y. and Chen, X. (2020). Explainable recommendation: A survey and new perspectives. *Foundations and Trends® in Information Retrieval*, 14(1):1–101.