Heterogeneous Graph Collaborative Filtering using Textual Information

Chaoyang Wang[a], Zhiqiang Guo[a], Guohui Li[b], Jianjun Li[a], Peng Pan[a], Ke Liu[a]

[a]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, 430074*
[b]*School of Software Engineering, Huazhong University of Science and Technology, Wuhan, China, 430074*

Abstract

Due to the development of graph neural network models, like graph convolutional network (GCN), graph-based representation learning methods have made great progress in recommender systems. However, the data sparsity is still a challenging problem that graph-based methods are confronted with. Recent works try to solve this problem by utilizing the side information. In this paper, we introduce easily accessible textual information to alleviate the negative effects of data sparsity. Specifically, to incorporate with rich textual knowledge, we utilize a pre-trained context-awareness natural language processing model to initialize the embeddings of text nodes. By a GCN-based node information propagation on the constructed heterogeneous graph, the embeddings of users and items can finally be enriched by the textual knowledge. The matching function used by most graph-based representation learning methods is the linear inner product, which cannot fit complex semantics well. We design a predictive network, which can combine the graph-based representation learning with the matching function learning, and demonstrate that this predictive architecture can gain significant improvements. Extensive experiments are conducted on three public datasets and the results verify the superior performance of our method over several baselines.

Keywords: Recommender system, Collaborative Filtering, Textual Information, Heterogeneous Graph, Graph Convolutional Network, Matching Function

1. Introduction

Due to the great expressive power of graphs, graph-based recommendation has received increasing attention in recent years [1–6]. Graph is a kind of data structure that models a set of objects (nodes) and their relationships (edges). Heterogeneous graphs, or heterographs for short, contain different types of nodes and edges. The graphs for recommendations are naturally heterographs as they contain at least two different types of

Email addresses: sunwardtree@hust.edu.cn (Chaoyang Wang), zhiqiangguo@hust.edu.cn (Zhiqiang Guo), guohuili@hust.edu.cn (Guohui Li), jianjunli@hust.edu.cn (Jianjun Li), panpeng@hust.edu.cn (Peng Pan), hustlk@hust.edu.cn (Ke Liu)

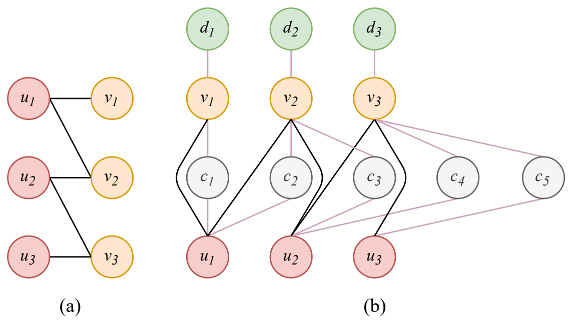


Figure 1: Graph Differences *w.r.t.* (a) the bipartite graph; (b) the heterograph with description and comment nodes.

nodes, i.e., user u and item v. To distinguish with other types of heterographs, the heterographs that only contain user-item interactions are a kind of bipartite graph, as depicted in Figure 1 (a). Recently, with the object of achieving more effective representations, some researchers try to exploit the bipartite graph structure by propagating user and item embeddings on it. For example, inspired by graph convolution network (GCN) [7], Wang *et al.* proposed NGCF [1] to derive more effective embeddings by following the same propagation rules as GCN, including feature transformation, neighborhood aggregation and non-linear activation. Later, in view that the nodes in the user-item interaction graph have no concrete semantics, both LR-GCCF [2] and LightGCN [3] found that discarding some components in GCN (such as non-linear activation and feature transformation) can yield better performance for collaborative filtering.

For other types of heterographs, existing works [5, 6, 8, 9] based on graph neural network (GNN) generally focus on leveraging side information. In this work, we construct heterograph by adding text nodes, including description d and comment c. As illustrated in Figure 1 (b), simply adding description and comment nodes cannot increase the indirect connections between users and items. Nevertheless, the text node itself contains plentiful knowledge. Before the learning process of the graph-based method, we can utilize advanced natural language processing (NLP) techniques, such as GloVe [10] and SBERT [11], to embed the text and then exploit these embeddings to initialize the text nodes, hence integrate with the semantics to enhance the representation ability. Moreover, adding these pre-trained embeddings of text nodes can also help alleviate the negative effects of data sparsity. Standard GCN is not suitable for the bipartite graph mainly comes from that the nodes in the user-item interaction graph do not contain concrete semantics [2, 3]. But in our constructed heterographs, both the description and comment nodes contain rich semantics. Therefore, we utilize the RGCN [12] (a modified version of standard GCN on heterograph) to learn more effective embeddings.

After deriving efficient user and item embeddings via feature propagation of GCN, another important component of the recommender system is the matching function. All the graph-based methods that learn the embeddings of users and items belong to representation learning. Almost all of the graph-based representation learning methods that exploit the bipartite graph utilize the inner product as the matching function [1, 3, 5, 6]. However, the simple linear inner product is ineffective for the non-linear features. To tackle the non-linear features learned by GNN from heterographs, GraphRec [8] concatenated latent factors of users and items and utilized an MLP [13] to predict ratings. Inspired by DeepCF [14], we design a predictive network that incorporates GCN-based representation learning with neural matching function learning to better fit the non-linear semantics. To sum up, the main contributions of this paper are as follows:

- We build a heterograph and initialize the embeddings of text nodes with GloVe or SBERT to exploit the knowledge in the textual information. Then, by utilizing RGCN, the semantics in these text nodes can be propagated to the representations of users and items.
- In view that the inner product is ineffective for the non-linear semantics, we propose a framework that can combine graph-based representation learning with neural matching function learning for better prediction.
- Extensive experiments are conducted on three benchmark datasets, the results verify the superior performance of our method over several baselines.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 details the proposed model; Section 4 presents and analyzes the experimental results; Finally, Section 5 concludes this paper.

2. Related Work

In this section, we review existing related works on graph-based, text-based and CF-based (CF, Collaborative Filtering) recommendation, respectively.

2.1. Graph-based Recommendation

As mentioned in the introduction, the heterographs can be divided into two categories: bipartite graphs and other heterographs, we discuss the graph-based recommendations in these two lines.

For bipartite graphs, early works, such as ItemRank [15] and BiRank [16], adopted the idea of label propagation to capture the CF effect, but the lack of model parameters for optimization limits their performance. Recently, HOP-Rec [17] incorporated matrix factorization (MF) and random walks, but it only used the graph-info to regularize to the MF and did not contribute to the embedding. GC-MC [18] applied a GCN-based auto-encoder framework on the bipartite user-item graph, but it only employed GCN for link prediction between users and items. Inspired by GCN, NGCF [1] exploited the collaborative signal in the embedding function and explicitly encodes the signal in the form of high-order connectivity by performing embedding propagation. The embedding propagation rule of NGCF is the same as that of GCN, which includes: feature transformation, neighborhood aggregation, and non-linear activation. But GCN is originally proposed for node classification on the attributed graph, where each node has rich attributes, whereas in the bipartite graph used by NGCF, each node is only described by a one-hot ID, which has no concrete semantics. By removing feature transformation and non-linear activation that will negatively increase the difficulty for the NGCF training phase, LightGCN [3] achieved significant accuracy improvements. However, by exploiting the side-information and keeping non-linear components, the performance can still be improved.

Different from bipartite graphs, the other heterographs exploited the side-information into the user-item interaction graph to enhance the recommendation ability. The researches on this domain mainly focus on the usage of knowledge graph (KG) [19]. Some works, such as CKE [20], DKN [21], RKGE [22], and KPRN [23], only exploited the knowledge of KG to enrich the representations [20, 21] or only leverage the connectivity patterns of the entity in KG to help the recommendation [22, 23]. To fully exploit the information in the KG, based on GNN, RippleNet [4] stimulated the propagation of user preferences by automatically and iteratively extending a user's potential interests along with links in the KG. KGCN [5] utilized GCN to effectively mine their representations which contains semantic information of KG and users' personalized interests in relations. When calculating the representation of a given entity in the KG, GCN-based KGCN aggregated and incorporated neighborhood information with bias. KGAT [6] refined the node's embeddings by propagating the embeddings from its neighbors in KG and employed an attention mechanism to discriminate the importance of the neighbors. For social recommendation, Fan *et al.* proposed the framework GraphRec [8] which coherently modeled social graph and user-item graph and heterogeneous strengths. Compared with constructing KGs, directly using easily available descriptions and comments is more convenient and low-cost. Liu *et al.* proposed HGNR [9] to learns the embeddings of users and items by using the GCN based on a heterograph, which was constructed from user-item interactions, social links, and semantic links predicted from the social network and textual reviews. However, HGNR just utilized the stander GCN but did not consider the difference between nodes. Therefore, the learning of heterograph was translated as a homogeneous process.

2.2. Text-based Recommendation

The applications of deep learning methods in NLP [10, 24, 25], make it possible to exploit textual information that human beings understand into recommender systems.

For the textual information in recommendations, the comments, which contain users' attitudes, and descriptions, which contain items' attributes, are important and can be collected easily. There are works that utilize sentiment analysis [26, 27], convolutional neural networks [28] and pre-trained word vectors on large corpora [24, 29] to get embeddings of comments and descriptions. The preferences of users contained in these embeddings can alleviate the negative effects of data sparsity, and thus can improve the performance of recommendations. However, the NLP techniques used by these text-based methods have weak context awareness, which may limit the ability to understand the textual information.

The appearance of the state-of-the-art context awareness NLP techniques, such as Transformer [30] and bidirectional encoder representations from transformers (BERT) [25], improves the performance of many NLP tasks significantly. Some researchers have applied these state-of-the-art pre-trained NLP models in citation recommendations [31, 32]. Hebatallah *et al.* [31] showed that the integration of traditional methods with pre-trained sentence encoders can retrieve more relevant papers. Chanwoo *et al.* [32] combined the context embeddings encoded by pre-trained BERT and the document embeddings encoded by GCN to learn the proper references for scientific papers.

Moreover, some researchers have tried to combine KG (a kind of relationship graph) embedding models with the textual information of entities. Richard *et al.* [33] introduced an expressive neural tensor network suitable for reasoning over relationships between two entities, and find that performance was improved when entities are represented as an average of their constituting word vectors. Xie *et al.* [34] proposed an RL method for KGs taking advantage of entity descriptions which is leaned by a continuous bag-of-words model and deep convolutional neural model. Xiao *et al.* [35] proposed the semantic space projection model which jointly learns from the symbolic triples and textual descriptions. Recently, HGNR [9] utilized SBERT [11] to explore the potential links between items based on comments.

2.3. CF-based Recommendation

Recently, DNNs which can capture non-linear user-item relationships and enable the codification of more complex abstractions have changed recommendation architecture dramatically [36]. Depending on the focus of the CF-based DNN methods, we can divide these methods into three categories [14]: representation learning, matching function learning, and the combination of them.

The main idea of representation learning for recommendation is mapping users and items into a common representation space where they can be compared directly. By using a two pathway neural network representation learning architecture, deep matrix factorization (DMF) [37] mapped the users and items into a common low-dimensional space with non-linear projections, and then utilized cosine similarity as the matching function to calculate predictive scores. Similar to DMF, to obtain matching scores, most of the graph-based representation learning methods [1–3, 5, 6] only utilized the inner product as the matching function. For these representation learning methods, the matching function used in the predictive part is too simple to deal with the non-linear features and can not fit the complex score distribution well.

Matching function learning is aiming to learn the complex matching function to model latent features by utilizing DNNs. He *et al.* [13] found that the linear inner product limits the ability to learn the complex structure of interactions and proposed MLP which replaces the inner product matching function with a non-linear neural architecture to learn the complex structure of user interaction data. By fusing the neural matching function learning method MLP with a representation learning method generalized matrix factorization, NeuMF was proposed to obtain better performance. Chen *et al.* proposed J-NCF [38] which applied a joint neural network that couples deep feature learning (representation learning) and deep interaction modeling (matching function learning) with a rating matrix. Further, considering the DNN-based representation learning and matching function learning suffered from two fundamental flaws, i.e., the limited expressiveness of inner product and the weakness in capturing low-rank relations respectively, Deng *et al.* proposed DeepCF [14], which combines the strengths of neural representation learning and neural matching function learning, to overcome such flaws. In this paper, inspired by the ideas of DeepCF, to better exploiting non-linear text features and learn complex matching function, we designed a new framework that can combine graph-based representation learning with neural matching learning.

3. Proposed Method

3.1. Framework Overview

We consider a recommender system with M users $\mathrm{U} = \{u_1, \dots, u_M\}$, N items $\mathrm{V} = \{v_1, \dots, v_N\}$, P descriptions $\mathrm{D} = \{d_1, \dots, d_P\}$ and Q comments $\mathrm{C} = \{c_1, \dots, c_Q\}$, and

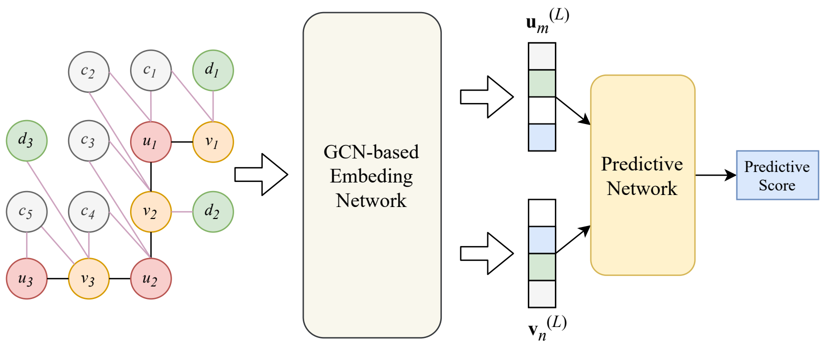


Figure 2: Overview of TextHGCF (An example of u_m and v_n).

the hidden state of them in l-th propagation layer are denoted as $\mathbf{U}^{(l)} = \{\mathbf{u}_1^{(l)}, \ldots, \mathbf{u}_M^{(l)}\}$, $\mathbf{V}^{(l)} = \{\mathbf{v}_1^{(l)}, \ldots, \mathbf{v}_N^{(l)}\}$, $\mathbf{D}^{(l)} = \{\mathbf{d}_1^{(l)}, \ldots, \mathbf{d}_P^{(l)}\}$ and $\mathbf{C}^{(l)} = \{\mathbf{c}_1^{(l)}, \ldots, \mathbf{c}_Q^{(l)}\}$ respectively. $Y \in \mathbb{R}^{M \times N}$ denotes the implicit feedback matrix, where $y_{m,n}$ is the implicit feedback of user u_m on item v_n. To avoid the negative influence of fake negative samples, we only utilize the interacted user-item pairs for train and test. Therefore, the value of $y_{m,n}$ can be defined as:

$$y_{m,n} = \begin{cases} 1, & \text{if } r_{m,n} \geq r_b; \\ 0, & \text{if } r_{m,n} < r_b. \end{cases} \tag{1}$$

where $r_{m,n}$ denotes the explicit rating of user u_m to item v_n, r_b is a boundary rating (e.g., $r_b = 3$ in a rating system with the highest rating 5).

Figure 2 gives an overview of the text-based heterogeneous graph collaborative filtering (TextHGCF) we propose. Firstly, in the heterograph, we initialize $\mathbf{u}_m^{(0)}$ and $\mathbf{v}_n^{(0)}$ randomly, and initialize $\mathbf{d}_p^{(0)}$ and $\mathbf{c}_q^{(0)}$ with existing NLP techniques. Additionally, to figure out the impact of context awareness, we utilize pre-trained GloVe and SBERT respectively. Then, on the GCN-based embedding network, by several embedding propagation layers, the textual knowledge in the text node embeddings can be integrated into the embeddings of users and items. Finally, the predictive network, which combines graph-based representation learning with neural matching function learning, takes $\mathbf{u}_m^{(L)}$ and $\mathbf{v}_n^{(L)}$ as input to get the predictive scores. In the training phase, we utilize pairwise BPR (bayesian personalized ranking) loss [39] to learn the parameters of the embedding network and the predictive network jointly.

3.2. Embeddings Initialization of Text Nodes

Textual information contains rich knowledge that can alleviate the negative effects of data sparsity. As mentioned above, we utilize two types of NLP techniques, GloVe [10] and SBERT [11], to initialize the embeddings of text nodes.

For the word to vector method GloVe, descriptions and comments contain many meaningless words that can affect the quality of embedding construction. Comparing with Long Stopword List[1], we remove them in advance. Then, we pick up pre-trained

[1]https://www.ranks.nl/stopwords

word vectors GloVe.6B[2] that has been trained on large corpora (Wikipedia 2014 and Gigaword 5) to calculate the initial $\mathbf{d}_p^{(0)}$ and $\mathbf{c}_q^{(0)}$. Specifically,

$$\mathbf{d}_p^{(0)} = \frac{1}{n_d}\sum\nolimits_{i=1}^{n_d} \mathbf{w}_i, \quad \mathbf{c}_q^{(0)} = \frac{1}{n_c}\sum\nolimits_{i=1}^{n_c} \mathbf{w}_i \tag{2}$$

where $\mathbf{w}_i$ denotes the vector of word w_i, n_d (n_c) denotes the number of words that d_p (c_q) contains after removing the stop words.

To get the independent context-awareness sentence embeddings, there are researches that passed sentences through BERT and then derive a fixed-sized vector by averaging the outputs. However, this common practice yields rather bad sentence embeddings, often worse than averaging GloVe embeddings [11]. To address this problem, Reimers *et al.* [11] proposed a context-awareness sentence embedding method SBERT, which adds a pooling operation after the pre-trained BERT [25] and utilizes siamese and triplet networks [40] to fine-tune the weights of BERT, and has been proved be capable of producing semantically meaningful sentence embeddings.

In our implementation, considering that the maximum dimension of pre-trained word vectors in GloVe.6B is 300, which is much smaller than the dimension of BERT-Base (768), we utilize the pre-trained BERT-Mini (256) [41] as the target fine-tune model. Moreover, since the MEAN strategy pooling operation (compute the mean of all output vectors) can always gain the best performance on several NLP tasks, the mean strategy is applied in the SBERT we implemented. Based on the pre-trained BERT-Mini and mean strategy pooling operation, SBERT can directly encode the sentences in d_p and c_q to initialize $\mathbf{d}_p^{(0)}$ and $\mathbf{c}_q^{(0)}$.

3.3. GCN-based Embedding Network

To utilize the knowledge in textual information, different from the original GCN that randomly initializes the embeddings of all the nodes, we initialize the embeddings of the text nodes in our heterograph by advanced NLP techniques. As shown in Figure 3, we take user u_1 as an example to concretely describe the first-order GCN-based embedding propagation, and then expand it to any nodes in high-order. Note that, since the edges in our heterograph only represent related associations, we do not learn and use the embeddings of them.

We utilize $e \in \mathrm{E}$ to denote the entities in heterograph, where $\mathrm{E} = \mathrm{U} \cup \mathrm{V} \cup \mathrm{D} \cup \mathrm{C}$, and use $\mathbf{e}$ to denote the representation of e. According to feature propagation rule of RGCN [12], we formulate the message-propagation for user u_1 under the relation set $\mathcal{R}$ as:

$$\mathbf{u}_1^{(1)} = a(\sum_{r\in\mathcal{R}} \sum_{m\in\mathcal{M}_{\mathrm{u}_1}^{\mathrm{r}}} g_m(\mathbf{u}_1^{(0)}, \mathbf{e}_j^{(0)})) \tag{3}$$

where $a(\cdot)$ is the activation function, $g_m(\cdot)$ is the message construction function, and $\mathcal{M}_{u_1}^r$ denotes the set of incoming messages for node u_1 under relation $r \in \mathcal{R}$. We utilize the non-linear LeakyReLU [42] as the activation function in this part.

Due to the introducing of textual information, the nodes in our heterograph are enriched with semantics. Therefore, we will utilize the standard GCN [7] to learn the

[2]http://nlp.stanford.edu/data/glove.6B.zip

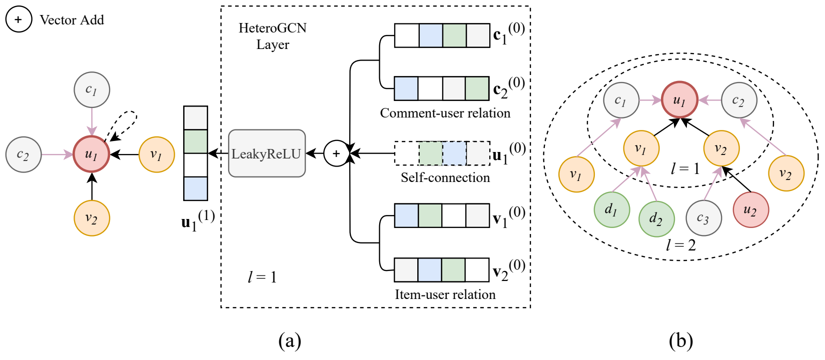


Figure 3: GCN-based embedding network *w.r.t.* (a) the 1st order GCN layer; (b) the 1st and 2nd order connectivity.

embeddings. In RGCN, $g_m(\cdot)$ is typically chosen to be a simple linear transformation, i.e., $g_m(\mathbf{e}_i, \mathbf{e}_j) = W\mathbf{e}_j$, where W denotes the weights. Note that, the weights are unique for different relation types. $\mathcal{M}_{u_1}^r$ is chosen to be identical the set of incoming edges under relation r, which contains $\mathcal{N}_{e_i}^r$ and the self-connection of u_1 in our method. $\mathcal{N}_{e_i}^r$ is denoted as the set of neighbor entities that directly connected to e_i under relation r. In this way, Equation (3) can be rewritten as:

$$\mathbf{u}_1^{(1)} = a\Big(\sum_{r\in\mathcal{R}}\sum_{e_j\in\mathcal{N}_{\mathrm{u}_1}^{\mathrm{r}}} \mathcal{L}_{u_1,e_j} W_j^{(0)}\mathbf{e}_j^{(0)} + W_{\text{self}}^{(0)}\mathbf{u}_1^{(0)}\Big) \tag{4}$$

where $\mathcal{L}_{u_1,e_j} = 1/\sqrt{|\mathcal{N}_{u_1}||\mathcal{N}_{e_j}|}$ is the symmetric normalization term that follows the design of RGCN, $W_j^{(0)}, W_{\text{self}}^{(0)} \in \mathbb{R}^{D_0\times D_1}$ are the trainable weights to distill useful information for propagation, and D_l is the transformation size of the l-th layer.

By propagating the hidden state through first-order connectivity, the embedding of u_1 can be obtained by Equation (4). To explore the high-order connectivity information, we should stack more embedding propagation layers. As Figure 3 (b) shows, when the number of the propagation layers l increases one, the nodes that can be used to calculate the embedding of u_1 increases four (d_1, d_2, c_3, u_2). Note that for an interacted user and item pair in our heterograph, there are two ways to connect them, one is the direct connection, the other is the indirect connection through the transmission of a comment. Therefore, the heterograph we used is a cyclic graph. In such a graph, the message propagation on some nodes will repeat, such as v_1 and v_2 in Figure 3 (b). But since GCN-based method can learn on cyclic graph, it has no impact on our implementation.

An entity e_i is capable to receive the messages propagated from its l-hop neighbors by stacking l embedding propagation layers, in the l-th step, the embedding of e_i is recursively formulated as:

$$\mathbf{e}_i^{(l)} = a\Big(\sum_{r\in\mathcal{R}}\sum_{e_j\in\mathcal{N}_{\mathrm{e}_i}^{\mathrm{r}}} \mathcal{L}_{e_i,e_j} W_j^{(l-1)}\mathbf{e}_j^{(l-1)} + W_{\text{self}}^{(l-1)}\mathbf{e}_i^{(l-1)}\Big) \tag{5}$$

where $\mathcal{L}_{e_i,e_j} = 1/\sqrt{|\mathcal{N}_{e_i}||\mathcal{N}_{e_j}|}$, $W_j^{(l-1)}$ and $W_{\text{self}}^{(l-1)} \in \mathbb{R}^{D_{l-1}\times D_l}$ are trainable weights,

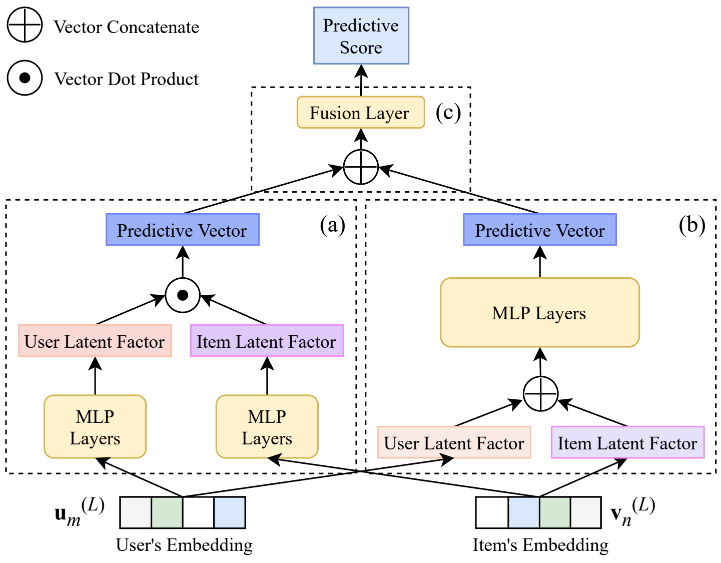


Figure 4: Predictive Network *w.r.t.* (a) representation learning part; (b) matching function learning part; (c) fusion part.

$\mathbf{e}_i^{(l-1)}$ and $\mathbf{e}_j^{(l-1)}$ respectively denote the representations of e_i and e_j generated from the previous message-propagation steps.

3.4. Predictive Network

After propagating with L layers, we can obtain $\mathbf{u}_m^{(L)}$ and $\mathbf{v}_n^{(L)}$ as the input of the predictive part. Different from the inner product used by most of graph-based representation learning methods, we design a predictive network which can incorporate graph-based representation learning with neural matching function learning.

As shown in Figure 4, our predictive network contains three part: (a) representation learning part, (b) matching function learning part, and (c) fusion part. Note for the GCN-based embedding network, itself belongs to representation learning, the representation learning part in our predictive network is mainly used to adjust the obtained embeddings. We adopt MLP (multi-layer perception) layers to transform the latent representations for users and items. Specifically, for user u_m,

$$\begin{aligned}\mathbf{h}_{u_m}^{(1)} &= a(\mathbf{W}_{rl}^{(1)}\mathbf{u}_m^{(L)} + \mathbf{b}_{rl}^{(1)}) \\ &\cdots\cdots \\ \mathbf{h}_{u_m}^{(X_{rl})} &= a(\mathbf{W}_{rl}^{(X_{rl})}\mathbf{h}_{u_m}^{(X_{rl}-1)} + \mathbf{b}_{rl}^{(X_{rl})})\end{aligned} \tag{6}$$

where $a(\cdot)$ denotes the activation function, $\mathbf{W}_{rl}^{(x_{rl})}$ and $\mathbf{b}_{rl}^{(x_{rl})}$ denote the weight matrix and bias vector for the x_{rl}-th layer's perception respectively. The activation function $a(\cdot)$ of the prediction network is LeakyReLU, which is also utilized in the embedding network. The latent representations $\mathbf{h}_{v_n}^{(X_{rl})}$ for item v_n can be calculated similarity.

Then, the latent predictive vector $\mathbf{h}_{rl}^{(X_{rl})}$ of representation learning is calculated by:

$$\mathbf{h}_{rl}^{(X_{rl})} = \mathbf{h}_{u_m}^{(X_{rl})} \odot \mathbf{h}_{v_n}^{(X_{rl})} \tag{7}$$

where $\odot$ denotes the vector dot product.

To better match the complex score distribution, matching function learning replaces the inner product with a neural network. By taking $\mathbf{u}_m^{(L)}$ and $\mathbf{v}_n^{(L)}$ as the input, we also adopt MLP layers to learn the matching function. Therefore, the calculation of latent predictive vector $\mathbf{h}_{ml}^{(X_{ml})}$ is formulated as:

$$\begin{aligned}
\mathbf{h}_{ml}^{(0)} &= \mathbf{u}_m^{(L)} \oplus \mathbf{v}_n^{(L)} \\
\mathbf{h}_{ml}^{(1)} &= a(\mathbf{W}_{ml}^{(1)}\mathbf{h}_{ml}^{(0)} + \mathbf{b}_{ml}^{(1)}) \\
&\cdots\cdots \\
\mathbf{h}_{ml}^{(X_{ml})} &= a(\mathbf{W}_{ml}^{(X_{ml})}\mathbf{h}_{ml}^{(X_{ml}-1)} + \mathbf{b}_{ml}^{(X_{ml})})
\end{aligned} \tag{8}$$

where $\oplus$ denotes the vector concatenate, $\mathbf{W}_{ml}^{(x_{ml})}$ and $\mathbf{b}_{ml}^{(x_{ml})}$ denote the weight matrix and bias vector for the x_{ml}-th layer, respectively.

The fusion part is utilized to aggregate the latent predictive vector of the above two parts and predict the score. In this part, the latent predictive vector $\mathbf{h}_{rl}^{(X_{rl})}$ and $\mathbf{h}_{ml}^{(X_{ml})}$ are aggregated by a vector concatenation. Finally, an MLP layer is applied as the mapping function to calculate the matching scores $\hat{S}$. Specifically, for user u_m and item v_n, the matching score $\hat{s}_{m,n}$ is calculated by:

$$\hat{s}_{m,n} = a(\mathbf{W}_{\text{out}}(\mathbf{h}_{rl}^{(X_{rl})} \oplus \mathbf{h}_{ml}^{(X_{ml})}) + \mathbf{b}_{\text{out}}) \tag{9}$$

where $\mathbf{W}_{out}$ and $\mathbf{b}_{out}$ respectively denote the weight matrix and bias vector of the fusion layer, $a(\cdot)$ is the LeakyReLU activation function here.

3.5. Optimization

The BPR loss, which has been intensively used in graph-based recommender systems [1–3], is utilized in our method to optimize the parameters of graph embedding network (θ^G) and predictive network (θ^P). The objective function is defined as:

$$\text{Loss} = \sum_{(m,i,j)\in \mathrm{O}} -\ln a\left(\hat{s}_{m,i} - \hat{s}_{m,j}\right) + \lambda\|\Theta\|_2^2 \tag{10}$$

where $\mathrm{O} = \{(m,i,j)|(u_m,v_i) \in \mathrm{R}^+, (u_m,v_j) \in \mathrm{R}^-\}$ denotes the pairwise training data, R^+ indicates the interactions that $y_{m,i} = 1$, and R^- indicates the interactions that $y_{m,j} = 0$; $a(\cdot)$ is the activation function and we use SoftPlus [43] here; Θ denotes all trainable model parameters, and λ controls the L2 regularization strength to prevent over-fitting. We adopt mini-batch Adam [44] to optimize the parameters of embedding network and predictive network jointly. The learning process of TextHGCF is presented in Algorithm 1.

Algorithm 1: Learning Algorithm of TextHGCF

```
Input: Entitie set of the heterograph: E, textual information set: T, number of
       embedding propagation layer: $L$, pairwise training data: O.
Output: Parameters of graph embedding network and predictive network: $\theta^G, \theta^P$.
1  Randomly initialize $\mathbf{U}^{(0)}$ and $\mathbf{V}^{(0)}$;
2  Initialize $\mathbf{D}^{(0)}$ and $\mathbf{C}^{(0)}$ with T by GloVe or SBERT;
3  while not converged do
4      for $l = 1$ to $L$ do
5          for $e_i$ in E do
6              Update $\mathbf{e}_i^{(l)}$ by Equation (5) and store it;
7          end
8      end
9      for $(u_m, v_i, v_j)$ in O do
10         Look up $\mathbf{u}_m^{(L)}$, $\mathbf{v}_i^{(L)}$, $\mathbf{v}_j^{(L)}$ from the embeddings of nodes;
11         Calculate $\hat{s}_{m,i}$, $\hat{s}_{m,j}$ by Equation (9) and store them;
12     end
13     Calculate BPR loss by Equation (10);
14     Utilize Adam to optimize $\theta^G$ and $\theta^P$ with BPR loss;
15 end
16 return $\theta^G$, $\theta^P$
```

4. Experiments

To demonstrate the effectiveness of the proposed method, we first introduce the experimental settings, and then present the experimental results to answer the following research questions:

- **RQ1**: How does our method perform as compared with other state-of-the-art CF methods?
- **RQ2**: How does the performance benefit from leveraging textual information and utilizing a non-linear matching function structure?
- **RQ3**: How do the key hyper-parameters (e.g., adding self connection or not, dropout of the predictive network, embedding size, depth of layer, etc.) affect the performance?

Note that, when analyzing one factor, we keep the others fixed. All the textual information embeddings are initialized with the pre-trained SBERT. The default settings are: the input embedding size is 256 (BERT-Mini) [41], the output embedding size is 64, the dropout rate of network is 0.4, the number of embedding propagation layer is 2, the depth of representation learning network is 1, the depth of matching function learning network is 2, the loss function is BPR loss. Our method is implemented based on Pytorch[3] and DGL[4], the codes will be released upon acceptance.

[3]https://github.com/pytorch/pytorch
[4]https://github.com/dmlc/dgl

Table 1: Statistics of datasets

Dataset	#User	#Item	#Pos Rating	#Neg Rating	Sparsity
Garden	1,686	962	12,080	1,192	99.1817%
Music	5,541	3,568	58,905	5,801	99.6727%
Baby	19,445	7,050	143,780	17,012	99.8827%

4.1. Experimental Settings

4.1.1. Datasets

Jure Leskovec *et al.* [45] collected and categorized a variety of *Amazon* products and built several datasets[5] including ratings, descriptions, and comments. We evaluate our models on three publicly available *Amazon* datasets: Patio Lawn and Garden (**Garden** for short), Digital Music (**Music** for short) and **Baby**, which all have at least 5 comments for each product. Among these datasets, from Garden to Baby, the size and the sparsity are increasing. Table 1 shows the statistical details of the datasets we used.

4.1.2. Baseline methods

We compare our method TextHGCF with seven methods, where singular value decomposition (SVD) is a basic linear CF recommendation method, DMF, MLP and DeepCF are neural recommendation methods that only utilize user-item interactions, NGCF and LightGCN are the state-of-the-art graph-based recommendation methods, ANR is a neural recommendation method that leverages textual information.

SVD [46], which is often used as a benchmark for recommendation tasks, aims to find the linear model that maximizes the log-likelihood of the rating matrix.

DMF [37] utilizes representation learning neural networks in matrix factorization, which can map the users and items into a common low-dimensional space with non-linear projections.

MLP [13] applies matching function learning neural networks in collaborative filtering, which utilizes non-linearity of DNNs for modeling user-item latent structures.

DeepCF [14] combines neural representation learning and neural matching function learning to overcome the limited expressiveness and the weakness in capturing low-rank relations.

NGCF [1] represents user-item interactions as a bipartite graph and explicitly encodes the collaborative signal in the form of high-order connectivity by performing embedding propagation based on graph convolutional network.

LightGCN [3] simplifies the design of GCN to learns embeddings by linearly propagating, which gains improvements over NGCF on the bipartite user-item interaction graph.

ANR [29] applies an attention mechanism to focus on the relevant parts of comments and estimates aspect-level user and item importance jointly with the aspect-based representation learning.

4.1.3. Evaluation Metrics and Methodology

In this method, we exploit the interacted user-item pairs to adjust the embeddings of graph nodes and parameters of the predictive network and evaluate the performance.

[5]http://snap.stanford.edu/data/amazon/productGraph/categoryFiles

Table 2: Performance comparison. The best performance is in boldface and the second is underlined, the percent of improvement is about the results between our method and ANR, the best one from existing methods.

Dataset	**Metric**	**Compared Methods**							**TextHGCF**
		SVD	**DMF**	**MLP**	**DeepCF**	**NGCF**	**LightGCN**	**ANR**	
Garden	MSE	0.49170	0.43967	0.43137	0.43439	0.41629	0.40045	0.36006	**0.32127**
	ACC	0.50830	0.56033	0.56863	0.56561	0.58371	0.59955	0.63994	**0.67873**
	AUC	0.50642	0.58394	0.60270	0.60577	0.61884	0.65379	0.67718	**0.72814**
Music	MSE	0.48918	0.39258	0.37651	0.37280	0.37558	0.37465	0.29222	**0.25842**
	ACC	0.51082	0.60742	0.62349	0.62720	0.62442	0.62535	0.70778	**0.74158**
	AUC	0.51057	0.65285	0.66412	0.67749	0.66576	0.68463	0.79525	**0.81779**
Baby	MSE	0.49484	0.43065	0.43140	0.42512	0.41591	0.40366	0.37191	**0.34955**
	ACC	0.50516	0.56891	0.56860	0.57489	0.58409	0.59634	0.62809	**0.65045**
	AUC	0.50891	0.59595	0.59516	0.60329	0.61021	0.62919	0.69407	**0.70879**

Based on the assumption that the scores of each user meet the Gaussian distribution (the assumption is for each user's preference, which is different from the long-tail distribution of all users in limited observations), we utilize Equation (11) to calculate the normalized score $\hat{S}^*$ and the predicted value $\hat{Y}$. For user u_m and item v_n, $\hat{s}^*_{m,n}$ and $\hat{y}_{m,n}$ can be calculated by:

$$\begin{aligned} \hat{s}^*_{m,n} &= (\hat{s}_{m,n} - \overline{s}_m)/\sigma_m \\ \hat{y}_{m,n} &= \begin{cases} 1, & \text{if } \hat{s}^*_{m,n} > 0; \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{11}$$

where $\overline{s}_m$ denotes the average predicted score of user u_m, σ_m denotes the variance of these scores. Then, mean squared error (MSE) can be calculated by y and $\hat{y}$ for testing. Based on the above transformation, we find that our model with pairwise BPR loss performs better than the one with regression MSE loss under regression MSE index. The details will be presented in Section 4.4. The criteria for evaluating a user's interest in an item is: whether $\hat{s}^*_{m,n}$ is greater than zero. In this way, we introduce the evaluation metrics of click-through rate [4, 5], ACC (accuracy) and AUC (area under the curve). ACC is calculated by y and $\hat{y}$, and AUC is calculated by y and $\hat{s}^*$. For MSE the smaller the better, but for ACC and AUC, the larger the better. Note that, to make ANR suitable for the 0/1 prediction, we also utilize this method to translate the predicted scores.

The test data was constructed in data preparation, and all the evaluated methods were tested by using this data. We now describe the test method in detail: For all the interacted user-item pairs, we first classify these pairs into positive and negative ones by comparing them with the boundary rating r_b. Then, we randomly selected 10% of interacted user-item pairs as testset (one half of the testset is from positive pairs and the other half is from negative pairs), the remaining 90% interacted user-item pairs are used as the trainset. Based on such a strategy, the recommendation methods can be evaluated by the metrics mentioned above.

4.2. Performance Comparison (RQ1)

Table 2 shows the summarized results of our experiments on the three datasets in terms of metrics including MSE, ACC, and AUC. From the results, we have the following key observations:

- Compared with the linear method SVD, the non-linear neural network methods perform better. Moreover, DeepCF which combines representation learning and matching function learning always performs better than DMF and MLP. Among these methods that only use user-item interaction information, graph-based LightGCN performs best.
- The structure of ANR is similar to that of DMF, but text-based method ANR outperforms DMF significantly, which demonstrates the importance of utilizing textual information to alleviate the negative effects of data sparsity.
- By exploiting pre-trained SBERT embeddings to initialize the text nodes of the heterograph and combining GCN-based representation learning with matching function learning to learn the preference of users, our TextHGCF obtains remarkable improvements over all the evaluated baselines.

4.3. Effect of Textual Information and Matching Function (RQ2)

To further figure out the impact of the textual information, in this part, we conduct experiments and discuss the results from different types of textual information and different methods for the initialization of text nodes in the heterograph.

Table 3: Types of textual information and initialization methods. Best performance is in boldface.

Dataset	**Metric**	**Textual Information Types**			**Initialization Methods**		**Default**
		W/O Text-info	**W/ Des.**	**W/ Com.**	**W/O Pre-train**	**W/ Glove**	
Garden	MSE	0.38688	0.35445	0.34993	0.36048	0.34917	**0.32127**
	ACC	0.61312	0.64555	0.65008	0.63952	0.65083	**0.67873**
	AUC	0.66121	0.70955	0.71456	0.69046	0.70610	**0.72814**
Music	MSE	0.28501	0.27852	0.26461	0.27790	0.27141	**0.25842**
	ACC	0.71499	0.72148	0.73539	0.72210	0.72859	**0.74158**
	AUC	0.78723	0.79399	0.81509	0.79824	0.81582	**0.81779**
Baby	MSE	0.37548	0.38270	0.35160	0.37741	0.35620	**0.34955**
	ACC	0.62452	0.61730	0.64840	0.62259	0.64380	**0.65045**
	AUC	0.67704	0.67295	0.70516	0.67115	0.70091	**0.70879**

As shown in Figure 1, the heterograph we used contains two types of textual information, descriptions and comments. Therefore, we have four cases to consider, without text-info, with descriptions, with comments and with all (the default). The comments contain rich knowledge of users and items, Table 3 shows that the addition of comments can always improve the performance. But adding descriptions only improves the performance little, sometimes even make the performance worse. This may be due to that the descriptions only contain the information that provided by the producers, which sometimes is not true. Compared with the situation that only exploits one kind of textual information, the combination of descriptions and comments achieves best performance. The right part of Table 3 shows that compared to the method without pre-training (randomly initialize the text nodes' embeddings), the methods that are initialized with advanced NLP techniques can enrich the knowledge of embeddings and thus gain a better performance. Further, compared with the simple word vector method GloVe, the context awareness method SBERT performs better.

Table 4: Effect of matching function of TextHGCF. Best performance is in boldface.

Dataset	**Metric**	**Inner Product**	**MLP**	**Default**
Garden	MSE	0.40573	0.32655	**0.32127**
	ACC	0.59427	0.67345	**0.67873**
	AUC	0.62761	0.72300	**0.72814**
Music	MSE	0.40896	**0.25688**	0.25842
	ACC	0.59104	**0.74312**	0.74158
	AUC	0.62726	0.81757	**0.81779**
Baby	MSE	0.43432	0.35315	**0.34955**
	ACC	0.56568	0.64685	**0.65045**
	AUC	0.58729	0.70089	**0.70879**

From Table 4, we can observe that, for our method, using non-linear matching function learning can achieve better performance, and the structure of our predictive network that combines graph-based representation learning with matching function learning can gain the best performance in most cases. Due to after introducing non-linear Semantics (textual information), compared with the simple inner product is a linear operation, the non-linear neural network can deal with the no-linear structure better and result in significant performance improvement.

4.4. Parameter Sensitivity (RQ3)

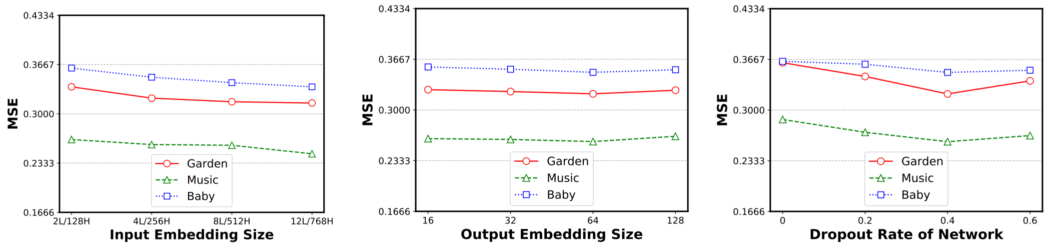
We select several important parameters (adding self-connection or not, types of loss function, input and output size of embedding network, dropout rate of the network, number of embedding propagation layer, depth of predictive network) to analyze their effects on the performance of our method. Since ACC and AUC exhibit similar performance trend as MSE, for simplicity, we only display the MSE of each dataset in the figures.

4.4.1. Adding self-connection or not.

After adding the self-connection (default), each node will include its own features in the process of summing the features of adjacent nodes. The results in Table 5 show that, with the help of self-connection, the performance of our model will improve substantially.

Table 5: If add self-connection and the loss types. Best performance is in boldface.

Dataset	**Metric**	**W/O Self-con.**	**MSE Loss**	**Default**
Garden	MSE	0.33937	0.33560	**0.32127**
	ACC	0.66063	0.66440	**0.67873**
	AUC	0.70450	0.72416	**0.72814**
Music	MSE	0.27249	0.27357	**0.25842**
	ACC	0.72751	0.72643	**0.74158**
	AUC	0.80052	0.80007	**0.81779**
Baby	MSE	0.37349	0.36230	**0.34955**
	ACC	0.62651	0.63770	**0.65045**
	AUC	0.68632	0.68863	**0.70879**



(a) (b) (c)

Figure 5: Performance of MSE *w.r.t.* (a) the input size of embedding network; (b) the output size of embedding network; (c) the dropout rate of network.

4.4.2. Types of loss function.

MSE is a regression loss whose target is to minimize the predictive scores with the ground-truth, BPR is a pairwise loss whose target is to enlarge the gap between positive samples and negative samples. For our target of distinguishing between positive and negative samples, BPR loss is more suitable. For the regression metrics, in general, the models trained by BPR will perform worse than that by MSE. However, as shown in the right part of Table 5, with the help of Equation (11), BPR (default) performs better in the three metrics.

4.4.3. Input and output size of embedding network.

Iulia *et al.* [41] release a couple of pre-trained miniature BERT models, which have smaller transformer layers (L) and hidden embedding sizes (H). The horizontal axis of Figure 5 (a) gives four combinations of L and H, from left to right are BERT-Tiny, BERT-Mini, BERT-Medium, and BERT-Base. It is easy to see that the more knowledge contained in the input embeddings, the better the performance.

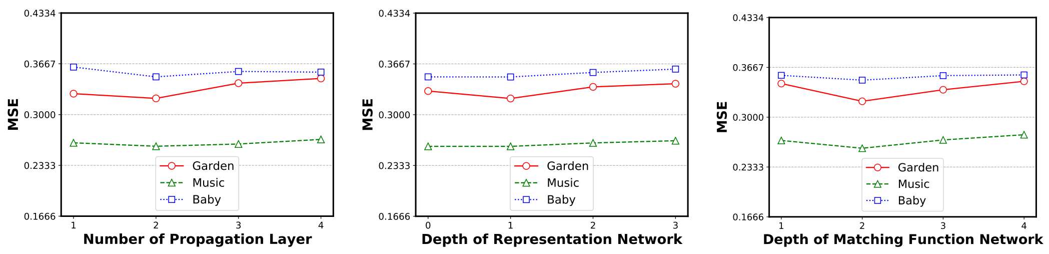
Figure 5 (b) shows that the performance of our method does not improve with the increase of the output size of the embedding network, it may be due to that the useful knowledge can be contained within 16 dimensions.

4.4.4. Dropout rate of network.

In Figure 5 (c), when the dropout rate ranges from 0 to 0.4, the performance of our model improves accordingly. However, the performance decreases when the dropout rate reaches 0.6. That is because the datasets we used are sparse, without dropout, the neural network is prone to over-fitting.

4.4.5. Number of embedding propagation layer.

Figure 6 (a) shows that with the increase of the embedding propagation layer number, the performance of our model first rises and then falls. It achieves the best performance when $l = 2$. The reason might be that the first-order embedding propagation in the graph carries limited information, while too high-order embedding propagation introduces noise, which will affect the model performance.



(a) (b) (c)

Figure 6: Performance of MSE *w.r.t.* (a) the number of propagation layer; (b) the depth of representation network; (c) the depth of matching function network..

4.4.6. Depth of predictive network.

As shown in Figure 4, in our predictive network, the depth of the fusion layer is fixed as 1. Figure 6 (b) and (c) show the performance of our method with varying depths of MLP layers for representation learning and matching function learning, respectively.

For the representation learning part, the best performance is achieved when the depth is 1. Representation learning is applied to get the proper representations of users and items, this work has been completed by the graph convolutional network, so only one layer to adjust the representations is enough. For the matching function learning part, proper depth of the network can improve the model learning ability, but too more layers will make the model training more difficult.

5. Conclusion

In this paper, to exploit the rich preference knowledge in textual information, we first build a heterograph and then utilize the pre-trained NLP models (GloVe or SBERT) to initialize the embeddings of these text nodes. Since the standard GCN is suitable for heterographs that contain concrete semantics, we apply RGCN (a modified version of standard GCN on heterograph) to propagate messages among neighbors, thus the knowledge in text nodes can be aggregated to the representations of users and items. Moreover, we find that the matching function used by most graph-based representation learning methods is the inner product, which is insufficient to process the non-linear semantics. Inspired by DeepCF, We propose a framework that can combine our graph-based representation learning method with the neural matching function learning, and train the GCN-based embedding network and predictive network jointly. We conduct extensive experiments on three public datasets, the results verify the superior performance of our TextHGCF model over several baselines. For future work, we intend to study if it is possible to incorporate our proposed method with reinforcement learning and transfer learning to meet the different needs of recommender systems.

References

[1] X. Wang, X. He, M. Wang, F. Feng, T.-S. Chua, Neural graph collaborative filtering, in: Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval, 2019, pp. 165–174.

[2] L. Chen, L. Wu, R. Hong, K. Zhang, M. Wang, Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach, arXiv preprint arXiv:2001.10167 (2020).

[3] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, M. Wang, Lightgcn: Simplifying and powering graph convolution network for recommendation, arXiv preprint arXiv:2002.02126 (2020).

[4] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, M. Guo, Ripplenet: Propagating user preferences on the knowledge graph for recommender systems (2018) 417–426.

[5] H. Wang, M. Zhao, X. Xie, W. Li, M. Guo, Knowledge graph convolutional networks for recommender systems, in: Proceedings of The 2019 Web Conference, 2019, p. 3307–3313.

[6] X. Wang, X. He, Y. Cao, M. Liu, T.-S. Chua, Kgat: Knowledge graph attention network for recommendation, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 950–958.

[7] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations (ICLR), 2017.

[8] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, D. Yin, Graph neural networks for social recommendation, in: The World Wide Web Conference, 2019, pp. 417–426.

[9] S. Liu, I. Ounis, C. Macdonald, Z. Meng, A heterogeneous graph neural model for cold-start recommendation, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, p. 2029–2032.

[10] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[11] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks (2019) 3973–3983.

[12] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, Springer, 2018, pp. 593–607.

[13] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2017, pp. 173–182.

[14] Z.-H. Deng, L. Huang, C.-D. Wang, J.-H. Lai, S. Y. Philip, Deepcf: A unified framework of representation learning and matching function learning in recommender system, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 61–68.

[15] M. Gori, A. Pucci, V. Roma, I. Siena, Itemrank: A random-walk based scoring algorithm for recommender engines., in: IJCAI, Vol. 7, 2007, pp. 2766–2771.

[16] X. He, M. Gao, M.-Y. Kan, D. Wang, Birank: Towards ranking on bipartite graphs, IEEE Transactions on Knowledge and Data Engineering 29 (1) (2016) 57–71.

[17] J.-H. Yang, C.-M. Chen, C.-J. Wang, M.-F. Tsai, Hop-rec: high-order proximity for implicit recommendation, in: Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 140–144.

[18] R. v. d. Berg, T. N. Kipf, M. Welling, Graph convolutional matrix completion, arXiv preprint arXiv:1706.02263 (2017).

[19] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, Q. He, A survey on knowledge graph-based recommender systems, arXiv preprint arXiv:2003.00911 (2020).

[20] F. Zhang, N. J. Yuan, D. Lian, X. Xie, W.-Y. Ma, Collaborative knowledge base embedding for recommender systems, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 353–362.

[21] H. Wang, F. Zhang, X. Xie, M. Guo, Dkn: Deep knowledge-aware network for news recommendation, in: Proceedings of the 2018 world wide web conference, 2018, pp. 1835–1844.

[22] Z. Sun, J. Yang, J. Zhang, A. Bozzon, L.-K. Huang, C. Xu, Recurrent knowledge graph embedding for effective recommendation, in: Proceedings of the 12th ACM Conference on Recommender Systems, 2018, pp. 297–305.

[23] X. Wang, D. Wang, C. Xu, X. He, Y. Cao, T.-S. Chua, Explainable reasoning over knowledge graphs for recommendation, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 5329–5336.

[24] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, Proceedings of Workshop at ICLR 2013 (01 2013).

[25] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.

[26] L. Qiu, S. Gao, W. Cheng, J. Guo, Aspect-based latent factor model by integrating ratings and reviews for recommender system, Knowledge-Based Systems 110 (2016) 233–243.

[27] K. Bauman, B. Liu, A. Tuzhilin, Aspect based recommendations: Recommending items with the most valuable aspects based on user reviews, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 717–725.

[28] D. Deng, L. Jing, J. Yu, S. Sun, H. Zhou, Neural gaussian mixture model for review-based rating prediction, in: Proceedings of the 12th ACM Conference on Recommender Systems, ACM, 2018, pp. 113–121.

[29] J. Y. Chin, K. Zhao, S. Joty, G. Cong, Anr: Aspect-based neural recommender, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 147–156.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.

[31] H. A. M. Hassan, G. Sansonetti, F. Gasparetti, A. Micarelli, J. Beel, Bert, elmo, use and infersent sentence encoders: The panacea for research-paper recommendation?, in: Proceedings of the 13th ACM Conference on Recommender Systems, Vol. 2431, 2019, pp. 6–10.

[32] C. Jeong, S. Jang, H. Shin, E. Park, S. Choi, A context-aware citation recommendation model with bert and graph convolutional networks, arXiv preprint arXiv:1903.06464 (2019).

[33] R. Socher, D. Chen, C. D. Manning, A. Ng, Reasoning with neural tensor networks for knowledge base completion, in: Advances in neural information processing systems, 2013, pp. 926–934.

[34] R. Xie, Z. Liu, J. Jia, H. Luan, M. Sun, Representation learning of knowledge graphs with entity descriptions, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[35] H. Xiao, M. Huang, L. Meng, X. Zhu, Ssp: semantic space projection for knowledge graph embedding with text descriptions, in: Thirty-First AAAI conference on artificial intelligence, 2017.

[36] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Computing Surveys (CSUR) 52 (1) (2019) 5.

[37] H.-J. Xue, X. Dai, J. Zhang, S. Huang, J. Chen, Deep matrix factorization models for recommender systems., in: IJCAI, 2017, pp. 3203–3209.

[38] W. Chen, F. Cai, H. Chen, M. D. Rijke, Joint neural collaborative filtering for recommender systems, ACM Transactions on Information Systems (TOIS) 37 (4) (2019) 1–30.

[39] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidtthieme, Bpr: Bayesian personalized ranking from implicit feedback (2009) 452–461.

[40] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 815–823.

[41] I. Turc, M.-W. Chang, K. Lee, K. Toutanova, Well-read students learn better: The impact of student initialization on knowledge distillation, arXiv preprint arXiv:1908.08962 (2019).

[42] B. Xu, N. Wang, T. Chen, M. Li, Empirical evaluation of rectified activations in convolutional network, arXiv preprint arXiv:1505.00853 (2015).

[43] C. Dugas, Y. Bengio, F. Belisle, C. Nadeau, R. Garcia, Incorporating second-order functional knowledge for better option pricing (2000) 472–478.

[44] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations (ICLR), 2015.

[45] J. Leskovec, A. Krevl, SNAP Datasets: Stanford large network dataset collection, `http://snap.stanford.edu/data` (Jun. 2014).

[46] S. Zhang, W. Wang, J. Ford, F. Makedon, J. Pearlman, Using singular value decomposition approximation for collaborative filtering, in: Seventh IEEE International Conference on E-Commerce Technology (CEC'05), IEEE, 2005, pp. 257–264.