Neural Computing and Applications (2022) 34:8211–8225
https://doi.org/10.1007/s00521-022-06926-x

ORIGINAL ARTICLE

Integrating label propagation with graph convolutional networks for recommendation

Yihao Zhang[1] · Meng Yuan[1] · Chu Zhao[1] · Mian Chen[1] · Xiaoyang Liu[2]

Received: 16 July 2021 / Accepted: 4 January 2022 / Published online: 23 January 2022

Abstract

Currently, graph convolutional networks (GCN) have achieved significant progress in recommender systems, due to its remarkable capability on representation learning and the ability to integrate complex auxiliary information. However, the graph convolution operation is prone to cause over-smoothing due to the use of the graph Laplacian operator, so that the node embeddings become very similar after the multi-layer graph convolution, which leads to a decrease in recommendation performance. The recently proposed model based on simplified GCN can relieve this issue to a certain extent; however, they still only design the model from the viewpoint of GCN. Inspired by the recent developments of label propagation algorithms (LPA), in this paper, we propose a new recommender model that unifies graph convolutional networks and label propagation algorithms. Specifically, we utilize the GCN to build a basic recommendation prediction model, and unify the LPA to provide regularization of training edge weights, which has been proven to effectively alleviate the over-smoothing problem. In addition, we introduce an attention network to capture the attention weight of each user-item pair, which takes into account the fact that users attach different degrees of importance to various relationships of items. Extensive experiments on three real-world datasets demonstrate that the proposed algorithm has a significant improvement over other state-of-the-art recommendation algorithms.

Keywords Recommender system · Over-smoothing problem · Graph convolutional networks · Label propagation algorithms

1 Introduction

Recommender Systems (RSs) play a key role in today's internet as they bring economic benefits to businesses while also offering personalized suggestions [1]. The collaborative filtering is the most widely used method in RSs [2], which produce new recommendations based on past interactions between users and items. This method only utilizes the implicit user feedback as input, and often suffers from inability of modeling auxiliary information (e.g., item attributes). In order to merge such information, a popular method is to convert them into feature vectors, and feed them into a supervised learning model with user-item ID. For example, factorization machine (FM) [3], NFM [4], and FNFM [5], etc.

Existing feature-based supervised learning methods treat each interaction as a separate data instance and fail to consider the high-order connectivity [6]. For example, the user u watched the movie v directed by the director

✉ Meng Yuan
yuanmeng123@2019.cqut.edu.cn

Yihao Zhang
yhzhang@cqut.edu.cn

Chu Zhao
zhaochu123@2019.cqut.edu.cn

Mian Chen
marian@2019.cqut.edu.cn

Xiaoyang Liu
lxy3103@cqut.edu.cn

[1] School of Artificial Intelligence, Chongqing University of Technology, Chongqing, China

[2] School of Computer Science and Engineering, Chongqing University of Technology, Chongqing, China

e. Collaborative filtering methods emphasize users who have also watched movie *v*, while feature-based supervised learning methods focus on the other movies directed by director *e*. In real situations, the identity of director *e* may also be an actor, screenwriter, etc., while the movies corresponding to such identities of *e* have not been explored by these methods.

To tackle this problem, GCN-based models [6–8] have achieved prominent progress due to the remarkable capability on representation learning, and the ability to integrate complex auxiliary information (e.g., social network and knowledge graph). The core idea of GCN-based method is to iteratively aggregate the neighbors of the target node and express it as the embedding of the target node, which has been proven to be an effective method for fully mining the high-order connectivity of the graph, thereby enriching the representation of users and items. For example, KGAT [6] leverages the attentive embedding propagation layers to measure the importance of information, and recursively mine high-order connected information. However, the graph convolution technique employs a type of graph Laplacian, which is easy to cause over-smoothing (i.e., the node representation becomes very similar after the multi-layer graph convolution) [7]. To put it simply, such operations will homogenize user preferences, decreasing recommendation performance. [9].

In order to solve the above limitations, there is an influx of recommender models based on simplified GCN [10–12]. For example, LightGCN [10] empirically demonstrates that feature transformations and nonlinear activations not only increase the difficulty of training, but also have no positive effect on the model. Similarly, LRGCN [11] has been empirically demonstrated that when more graph convolution layers are stacked, the model's performance degrades. Therefore, the above models usually solve these problems by reducing the number of network layers or removing nonlinear transformations; however, they still only design models from the perspective of graph convolution. As we know, GCN propagates and transforms node feature information along the edges of the graph [13], while label propagation algorithm (LPA) [14] propagates node label information. Therefore, models based solely on GCN are prone to be limited by the feature influences, and lack node information from other perspectives.

To solve these problems, we investigate the theoretical relationship between GCN and LPA, and propose a new recommender model named GCNLP. Different from existing GCN-based models, we integrate LPA to assist GCN-based models to alleviate the over-smoothing problem easily caused by GCN. Specially, we utilize GCN for basic prediction, and LPA provides regularization of training edge weights. In addition, we introduce an attention network to capture the attention weight of each user-item pair, which considers the fact that users assign different degrees of importance to various relationships of items. To evaluate the effectiveness of our proposed model, we conduct comprehensive experiments to compare our model with several baselines on Movielens-1M dataset, LastFM dataset and Book-Crossing dataset. Experiment results show that our algorithm outperforms the state-of-the-art recommendation algorithm. To sum up, the main contributions of this paper are as follows.

(1) We propose a novel algorithm integrating label propagation with graph convolutional networks for recommendation. Leveraging LPA assist GCN to regularize edge weight, which can effectively alleviate the over-smoothing problem.
(2) We introduce an attention network to capture the attention weight of each user-item pair, which takes into account the fact that users attach different degrees of importance to the various relationships of items.
(3) We perform extensive experiments on three public datasets to prove the superiority of our proposed recommender method. Experimental results show that our algorithm outperforms state-of-the-art algorithms.

2 Related work

Existing recommender models incorporated with knowledge graph (KG) can be roughly divided into three types, embedding-based methods, path-based methods and GCN-based methods.

Embedding-based methods usually apply knowledge graph embedding (KGE) algorithms to encode the knowledge graph to represent items, and then integrate the item auxiliary information into the recommendation framework [15]. For instance, Zhang et al. [16] designed the CKE recommendation model, which applies TransE [17] on KG triplets, and feed the knowledge-aware embedding of items into matrix factorization (MF) [18]. In order to further capture users' dynamic interests, Wang et al. [19] proposed the DKN model by integrating entity embedding and word embedding together with the CNN [20] framework for news recommendation, which also aggregates the embedding of historical clicked sequence to learn users representation. Subsequently, Wang et al. [21] proposed the SHINE algorithm, which treats the recommendation task as link prediction between entities. Another trend that follows this strategy is to adopt multi-task joint learning. For example, Cao et al. [22] designed the KTUP model, which simultaneously complete the task of recommendation and knowledge graph completion.

Path-based methods investigate various connections among items in knowledge graph to provide additional guidance for recommendation systems. For example, Yu et al. [23] proposed the HeteMF model, which extracts different meta-paths and computes item-item similarities in each path, then refines representation of uses and items. Luo et al. [24] proposed the HeteCF model by further considering the user-user similarity and user-item similarity. Yu et al. [25] designed the HeteRec model, which directly uses the meta-path similarity to enhance the user-item interaction matrix, therefore users/items can be represented more comprehensively. Subsequently, Yu et al. [26] designed the HeteRec model, which takes into account that the relevance of distinct paths depending on each user. To break through the limitation of meta-path expression ability, Zhao et al. [27] proposed the FMG model by substituting the meta-graph for the meta-path, which contains richer connectivity information than meta-path and captures the similarity between entities more accurately. In order to reduce the number of meta-path selections, Ma et al. [28] proposed the RuleRec model, which learns relations among connected items by exploring items' connectivity.

GCN-based methods mainly focus on the information aggregation mechanism. Specifically, it incorporates information from the neighbor nodes to update the representations of ego nodes; When such propagation is performing recursively, information from the multi-hop nodes can be encoded in the embedding. Therefore, these methods are able to model high-order connectivity. For example, KGAT [6] connects user-item interactions and KG as a heterogeneous graph, and then employs the aggregation mechanism on it. Recently, inspired by the recent theories in simplifying GCNs [12], He et al. [10] verified that feature transformations and nonlinear activations of GCN have negative effects on recommendation performance, and proposed LightGCN that achieves the SOTA performance. Recently, CKAN [29] leverages two different strategies to propagate collaborative signals and knowledge awareness signals, respectively. And NIRec [30] is designed to merge path-based and GCN-based models, which propagate interaction patterns between two nodes through neighborhoods guided by meta-paths.

To sum up, embedding-based methods apply KGE methods to preprocess the KG for obtaining the embeddings of entities, and further integrate them into the recommendation framework. However, the informative connectivity of the KG is ignored in these methods, and few of them can explain the recommendation results. On the other hand, path-based methods utilize KG in a more natural and direct way, but they need massive manually designed meta-paths, which are very labor-intensive and time-consuming. GCN-based methods benefit from both the KGE techniques and the semantic path pattern. However, models based solely on GCN are prone to cause over-smoothing. Although some latest solutions can alleviate the problem by simplifying GCN, we still deem that they lack node information from other perspectives. We argue that integrating LPA into GCN can alleviate the over-smoothing. Therefore, we propose a new recommender model integrating LPA with GCN.

3 Method

We first introduce the recommendation problem, formulate the label propagate algorithm and graph convolutional network; then in the message propagation process, we explore the relationship between GCN and LPA, and prove the necessity of graph convolution operation to improve recommendation performance. Finally, according to the above theoretical guidance, we integrate LPA with GCN to design a new recommender model.

3.1 Problem formulation

The problem of recommender systems based on knowledge graphs is described as follows. Let $U = \{u_1, u_2, \ldots\}$ and $V = \{v_1, v_2, \ldots\}$ denote the sets of users and items, respectively. The user-item interaction matrix $Y = \{y_{uv} | u \in U, v \in V\}$ is defined based on the users' implicit feedback. We define the user–item interaction matrix as a

$$y_{uv} = \begin{cases} 1, & \text{if interaction } (u, v) \text{ is observed;} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

here the value of 1 for y_{uv} denotes that there is an interaction between user u and item i; otherwise $y_{uv} = 0$. And we have a knowledge graph G available, which consists of massive entity-relation-entity triplets (h, r, t), where $h \in \phi$, $r \in \varphi$, and $t \in \phi$ denote the head, relation, and tail of knowledge triple, respectively; ϕ and φ denote the set of entities and relations in G, respectively. Our goal is to predict the probability of each user clicking on any items. For example, in the movie recommendation scene, the problem formulation is shown in Fig. 1.

3.2 Label propagation algorithm

Label Propagation Algorithm (LPA) is based on the assumption that connected nodes are more likely to have similar labels. For a graph $G = (V, A, X, P)$, where $V = \{v_1, v_2, .., v_n\}$ denotes the set of nodes, A is the adjacency matrix, a_{ij} (the ij-th entry of A) represents the weight of the edge between v_i and v_j. $N(v)$ is the set of direct neighbors of node v. P denotes the feature matrix of nodes and X is

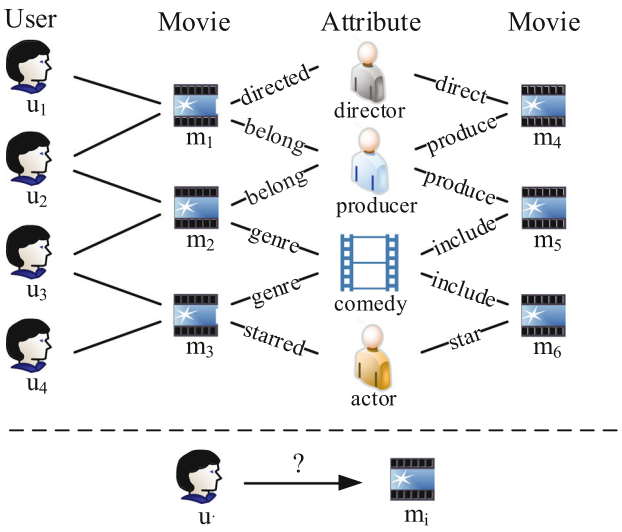


Fig. 1 Top: A small example knowledge graph. Bottom: The learning objective is predicting the probability of u_i clicking on m_i

labels of nodes, in more detail, $X^{(k)} = [{x_1}^{(k)}, \ldots, {x_n}^{(k)}]^T$ as the label matrix, where the i-th row ${x_i}^{(k)T}$ represents the predicted label of node v_i in iteration k. The initial label matrix $X^{(0)} = [{x_1}^{(0)}, \ldots, {x_n}^{(0)}]^T$ denotes one-hot label vectors (if node is labeled) or zero vectors (if node is not labeled). D is the diagonal degree matrix for A. And m represents the number of labeled nodes. Then LPA in step k is expressed as Eq. (2):

$$\begin{aligned} X^{(k+1)} &= D^{(-1)}AX^{(k)}, \\ {x_i}^{(k+1)} &= {x_i}^{(0)}, \quad \forall i \leq m. \end{aligned} \tag{2}$$

$${x_i}^{\infty} = \sum_{j \in N(i)} a_{ij} {x_j}^{\infty}. \tag{3}$$

Equation (3) indicates that the final representation of the node is the weighted average of its neighbor nodes. In addition, Fig. 2 shows a demo of label propagation process.

3.3 Graph convolutional network

Graph convolutional network (GCN) is a multi-layer feedforward network, and it can propagates and transforms node features in the graph. Which can be formulated as: $P^{(k+1)} = \sigma(D^{-1/2}AD^{-1/2}P^k W^{(k)})$, where $W^{(k)}$ denotes weight matrix in the k-th layer, $\sigma()$ represents an activation function, and $P^{(k)} = [{p_1}^{(k)}, \ldots, {p_n}^{(k)}]$ are the k-th layer node representations with $P^{(0)} = P$. To be consistent with LPA, we treat $D^{(-1)}A$ as the normalized adjacency matrix instead of $D^{-1/2}AD^{-1/2}$. Thus, the feature propagation of GCN in the layer k can be formulated as:

$$P^{(k+1)} = \sigma(D^{-1}AP^k W^{(k)}). \tag{4}$$

Similar to LPA, the target node representation is the weighted sum of its neighbor nodes:

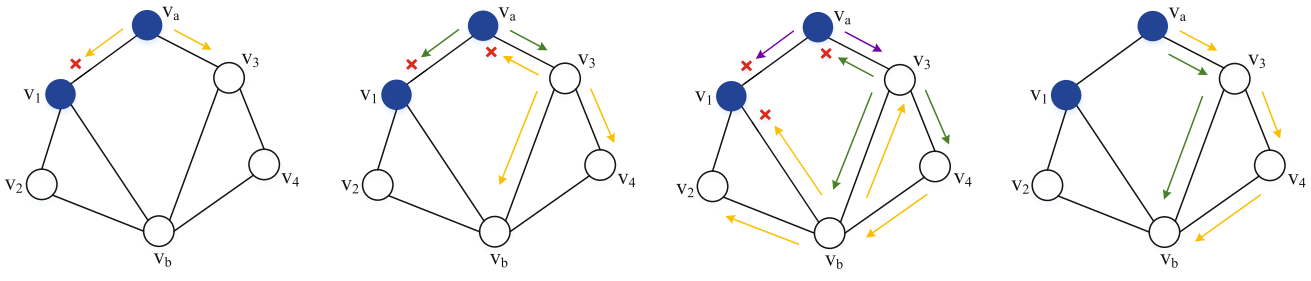
$${p_i}^{\infty} = \sum_{j \in N(i)} a_{ij} {p_j}^{\infty} \tag{5}$$

3.4 Relationship between LPA and GCN

Assume that two nodes v_1 and v_2 in a graph, v_2 is labeled and v_1 is unlabeled, we investigate the relationship between GCN and LPA from the perspective of influence. In particular, if the initial feature/label appears, how the output feature/label of v_1 will change, and v_2 will slightly change. Technically speaking, the feature/label influence is measured by the Jacobian/gradient of the output feature/label of v_1 with respect to the initial feature/label of v_2 [13]. ${p_1}^{(k)}$ denotes the k-th layer embedding of v_1 in GCN, and p_2 represents the initial embedding of v_2. We calculate the feature influence of v_2 on v_1 as follows:

$$I_f(v_1, v_2; k) = ||E[\partial {p_1}^{(k)} / \partial p_2]||. \tag{6}$$

On the other hand, we compute the label influence of node v_2 on node v_1 in LPA as Eq. (7). Since LPA is an iterative



(a) Step 1 **(b)** Step 2 **(c)** Step 3 **(d)** Paths from v_a to v_b

Fig. 2 A demo of label propagation process. Assume that labels are propagated for three steps. The blue nodes mean labeled, while the white nodes mean unlabeled. Information can only be propagated from labeled nodes to unlabeled nodes, and then the starting node v_a is reset to its initial value for the next propagation, finally we can count all possible paths from node v_a to node v_b

algorithm, the influence of x_2 on x_1 is the cumulative influence as Eq. (8).

$$I_l(v_1, v_2; k) = \partial {x_1}^{(k)} / \partial x_2 \tag{7}$$

$$I_l(v_1, v_2; k) = \sum_{j=0}^{k-1} \partial {x_1}^{(k)} / \partial {x_2}^{(j)} \tag{8}$$

We set ReLU as the activation function of GCN, and β denotes the fraction of node. Finally, we get that the feature influence and label influence satisfy the following equation:

$$E[I_l(v_1, v_2; k)] = \sum_{j=1}^{k} \beta^j \tilde{I}_f(v_1, v_2; j). \tag{9}$$

The details of the proof of this equation can be referred to paper [31]. And in Eq. (9), $\tilde{I}_f(v_1, v_2; j)$ is the normalize feature influence, which can be computed as:

$$\tilde{I}_f(v_1, v_2; k) = \frac{I_f(v_1, v_2; k)}{\sum_{v_i \in V} I_f(v_1, v_i; k)}, \tag{10}$$

Equation (9) proves that if v_2 has a higher label influence on v_1, the initial feature vector of v_2 will also affect the output feature vector of v_1 to a large extent. Intuitively, Eq. (9) indicates that when feature influence is required, label influence can be used instead. Therefore, LPA has the potential to replace GCN training edge weights.

3.5 Effect of graph convolution operation

For node v_i, the update formula of GCN as follows:

$$p_i^{(k+1)} = \sigma\left(\sum_{v_j \in N(v_i)} a_{ij} p_j^{(k)} W^{(k)} \right). \tag{11}$$

Equation (11) can be decomposed into the following two steps:

(1) In aggregation step, we compute the aggregated embedding $s_i^{(k)}$ of neighborhoods $N(v_i)$:

$$s_i^{(k)} = \sum_{v_j \in N(v_i)} a_{ij} p_j^{(k)}. \tag{12}$$

(2) In transformation step, the aggregated embedding $s_i^{(k)}$ is converted to the final representation:

$$p_i^{(k+1)} = \sigma(s_i^{(k)} W^{(k)}). \tag{13}$$

We prove that through the aggregation step, the distance between the connected nodes in the embedding space can be reduced. We define $Q(x) = \frac{1}{2} \sum_{v_i, v_j} a_{ij} ||p_i - p_j||_2^2$ to be the distance metric between v_i and v_j. Finally, we will have

$$Q(s^{(k)}) \le Q(p^{(k)}). \tag{14}$$

The proof process is as follows. Firstly, we compute the Hessian of $Q(p)$ as:

$$\begin{aligned} \nabla^2 Q(p) &= \begin{bmatrix} 1 - a_{11} & -a_{12} & \cdots & -a_{1n} \\ -a_{21} & 1 - a_{22} & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & 1 - a_{nn} \end{bmatrix} \\ &= E - Q^{-1} A \end{aligned} \tag{15}$$

It's obvious that $2E - \nabla^2 Q(x) = E + Q^{-1}A$. Since $Q^{-1}A$ is Markove matrix, its eigenvalues are within the range $[-1, 1]$, and the eigenvalues of $E + Q^{-1}A$ are within the range [0, 2]. Therefore, $E + Q^{-1}A$ is a positive semi-definite matrix, we have $\nabla^2 Q(x) \le 2E$. Then we can also deduce that

$$\begin{aligned} Q(s^{(k)}) &= Q(p^{(k)}) + \nabla Q(p^{(k)})^\top (s^{(k)} - p^{(k)}) \\ &\quad + \frac{1}{2} (s^{(k)} - p^{(k)})^\top \nabla^2 Q(x) (s^{(k)} - p^{(k)}) \\ &= Q(p^{(k)}) - \nabla Q(p^{(k)})^\top \nabla Q(p^{(k)}) \\ &\quad + \frac{1}{2} \nabla Q(p^{(k)})^\top \nabla^2 Q(p) \nabla Q(p^{(k)}) \\ &\le Q(p^{(k)}) - \nabla Q(p^{(k)})^\top \nabla Q(p^{(k)}) \\ &\quad + \nabla Q(p^{(k)})^\top \nabla Q(p^{(k)}) \\ &= Q(p^{(k)}). \end{aligned} \tag{16}$$

Equation (16) states that after an aggregation step, the total distance between connected nodes is reduced. In other words, the aggregation process combines together nodes that belong to the same class. As a result, GCN potentially categorize items that users are interested in into the same category, assisting the model in improving recommendation performance.

3.6 Unified model

We present the overall framework of GCNLP model, shown in Fig. 3. Firstly, we utilize the KG triples of $< user, item, relation >$ as the input, and assign them with initial embeddings. Secondly, we utilize the inner product of user embeddings and relation embeddings to indicate the importance of the relationship to each user, select neighbors for the target node according to the attention weight, and transform the KG into a weighted graph (this step is called attention layer in Fig. 3). Thirdly, we feed the weighted graph into GCN for training to generate new entity embeddings as item embeddings. At the same time, we perform label propagation on the weighted graph, get another item embeddings. Finally, we leverage the inner product of user embeddings and item embeddings to denote

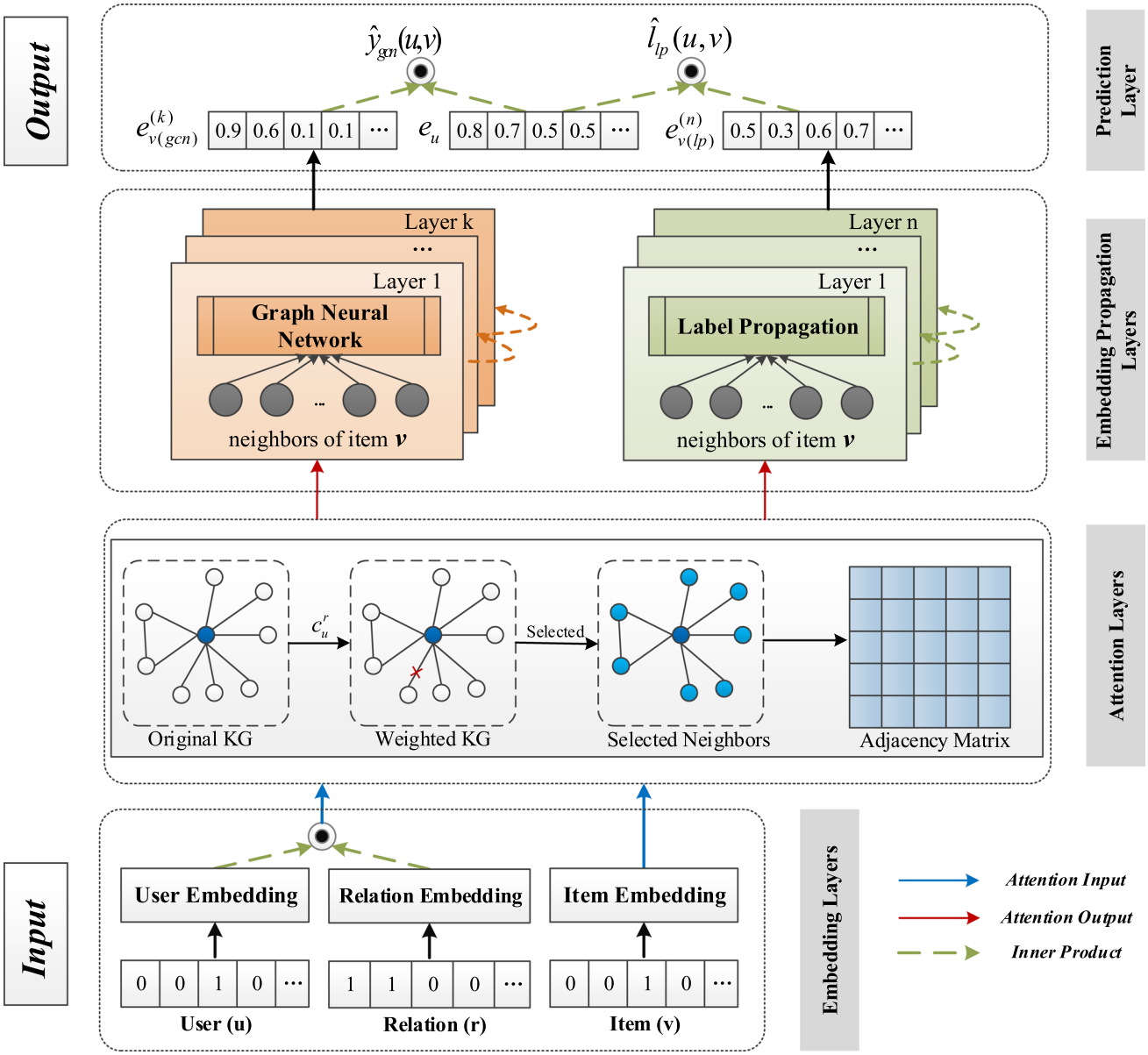


Fig. 3 An illustration of GCNLP model architecture. The first step is to select neighbors for the target node according to the attention weight, and transform the KG into a weighted graph. Secondly, we feed the weighted graph into GCN for training to generate a new entity embeddings as item embeddings. At the same time, we perform label propagation on the weighted graph, get another item embeddings. Finally, we leverage the inner product of user embeddings and item embeddings to denote the final prediction

the final prediction. We will describe the architecture in detail in the following subsections.

The early phase of the method is to convert the heterogeneous KG into weighted graphs to characterize users preferences. To this purpose, we introduce a user-specific relation-aware attention scoring function c^r_u, which provides the user u with the importance of relation r. According to the formula:

$$c^{\mathbf{r}}_{\mathbf{u}} = \mathbf{u}^{\mathrm{T}}\mathbf{r}, \tag{17}$$

where $\mathbf{u}$ and $\mathbf{r}$ denote the user embeddings and relation embeddings. Intuitively, c^r_u denotes the importance of relation r to user u. The original knowledge graph is an unweighted graph, where the edges only represent the relationship type, and the weights are not displayed. After introducing c^r_u, the original knowledge graph will be converted into a weighted graph. In the real scene, the scale of the weighted graph is usually very large. To alleviate the computational burden, we introduce receptive fields to determine the number of nodes. In particular, we uniformly sample a set of neighbors of a fixed-size k for each entity, where the priority of neighbor selection always choose

fixed k neighbors of the highest weights. The neighborhood of v is defined as

$$v_{N(v)}^{u} = \sum_{\mathbf{e} \in N(v)} \hat{c}_{\mathbf{u}}^{\mathbf{r}} \mathbf{e}, \tag{18}$$

where $\mathbf{e}$ denotes embedding of entity e, $N(v)$ denotes the selected neighbor set of v, and $\hat{c}_u^r$ is the normalized user-relation score as

$$\hat{c}_{\mathbf{u}}^{\mathbf{r}} = \frac{exp(c_{\mathbf{u}}^{\mathbf{r}})}{\sum_{e \in N(v)} exp(c_{\mathbf{u}}^{\mathbf{r}})}, \tag{19}$$

We can also treat the weighted graph as an adjacency matrix A_u, where the (i, j)-entry $A_u{}^{ij} = c_u(r_{e_i,e_j})$, and r_{e_i,e_j} represents the relation between entities e_i and e_j in KG. We define the original feature matrix of entities as $E \in R^{|\phi| \times |d_0|}$, where d_0 is the dimension of original entity features. Then, we leverage multiple network layers to update the entity representation. In particular, the layer-wise propagation can be formulated as

$$\begin{aligned} H_1 &= \sigma(D_u^{-1/2} A_u D_u^{-1/2} H_0 W_0) \\ H_2 &= \sigma(D_u^{-1/2} A_u D_u^{-1/2} H_1 W_1) \\ &\cdots \\ H_l &= \sigma(D_u^{-1/2} A_u D_u^{-1/2} H_{l-1} W_{l-1}), \end{aligned} \tag{20}$$

where H_l denotes the matrix of representations of entities in layer l, each row is the embedding of each item denoted as h_l, and $H_0 = E$, A_u represents the weighted sub-graph mentioned above. D_u is a diagonal degree matrix with entries $D_u^{ii} = \sum_j A_u^{ij}$, therefore, $D_u^{-1/2}$ is used to normalize A_u and maintain the stability of entity representation matrix H_l. $W_l \in R^{d_l \times d_{l+1}}$ denotes the layer-specific weight matrix, σ is a nonlinear activation function, and l represents the number of layers. The GCN prediction is defined as the inner product of user and the final representation of items:

$$\hat{y}_{uv} = u^{\mathrm{T}} h_l, \tag{21}$$

where u represents the user embedding, which is obtained from the user id through the embedding lookup layer, and $\hat{y}_{uv}$ represents the ranking score for recommendation generation.

Unlike traditional GCN-based methods, which take edge weights of the input data as constants, edge weights $D_u^{-1/2} A_u D_u^{-1/2}$ in Eq. (20) are trainable in our model since we introduce the user-relation score function Eq. (17). Though this operation strengthens the model's fitting power, it also makes the optimization process increasingly prone to cause over-smoothing, because the sole source of supervised signal is user-item interactions. Furthermore, edge weights play a crucial role in graph representation learning. As a result, more regularization on edge weights is required to facilitate the training of entity representations, and more likely to seek items that interest users. We demonstrate that LPA has the potential to replace GCN training edge weights in section 3.4. Therefore, we update A_u with the LPA algorithm, and use it to regularize edge weights for better performance.

Now we take out an item v and remove its label, and predict the unmarked item by using the remaining labeled items. The prediction process is shown as Eq. (2), the Label Propagation prediction is also defined as the inner product of user and item:

$$\hat{l}_{uv} = u^{\mathrm{T}} y, \tag{22}$$

then we update the edge weight of A_u by reducing the difference between the label y_{uv} of the real item v and the predicted label $\hat{l}_{uv}$.

The formulation of the above steps as shown in Algorithm 1. H represents the depth of receptive field. For each user-item pair (line 2), we compute the receptive field M of v (line 3, 21-29), and obtain the neighborhood representation(line 4). Then we feed them into GCN (line 5) and finally get $\hat{y}_{uv}$ (line 10). The calculation method of LPA is similar to GCN, and finally obtain $\hat{l}_{uv}$ (line 20). Unifying GCN and LPA, we get the following complete loss function:

$$loss = \sum_{u,v} J(y_{uv}, \hat{y}_{uv}) + \lambda \left(\sum_{u,v} J(y_{uv}, \hat{l}_{uv}) \right) + \gamma \|\theta\|_2^2, \tag{23}$$

where J is the cross-entropy loss function, $\|\theta\|_2^2$ is the L2-regularizer, λ and γ are balancing hyper-parameters. In Eq. (23), the first term corresponds the part of GCN that learns the edge weights A and the trainable matrix W. While the second term corresponds the part of label propagation that can be treated as adding constraint on edge weights A. We can regard the second term as a regularization on A to assist GCN in learning edge weights.

Algorithm 1: GCNLP Algorithm

Input: Interaction matrix Y; knowledge graph $G(\phi, \varphi)$; neighbor sampling mapping S
Output: The GCN prediction $\hat{y}_{uv}$, LPA prediction $\hat{l}_{uv}$

while *GCN not converge* **do**
  for *(u, v) in Y* **do**
    $M[i]_{i=0}^{H} \leftarrow$ Get-Receptive-Field(v);
    $v_{N(v)}^{u} \leftarrow e, \forall e \in N(v)$;
    Update entity representation by Eq.(20);
    Calculate $\hat{y}_{uv}$ according to Eq.(21) ;
    Update parameters by gradient descent ;
  end
end
return $\hat{y}_{uv}$;
while *LPA not converge* **do**
  for *(u, v) in Y* **do**
    $M[i]_{i=0}^{H} \leftarrow$ Get-Receptive-Field(v);
    $v_{N(v)}^{u} \leftarrow e, \forall e \in N(v)$;
    Update entity representation by Eq.(2);
    Calculate $\hat{l}_{uv}$ according to Eq.(22) ;
    Update parameters by gradient descent ;
  end
end
return $\hat{l}_{uv}$;
Function *Get-Receptive-Field(v)*:
  $M[h] \leftarrow e$;
  for *h = H - 1, ..., 0* **do**
    $M[h] \leftarrow M[h+1]$;
    for *e in M[h+1]* **do**
      $M[h] \leftarrow M[H] \cup S(e)$;
    end
  end
return $M[i]_{i=0}^{H}$

3.7 Time complexity analysis

As can be seen, the layer-wise propagation is the core operation. For the l-th layer, the computational complexity of matrix multiplication is $O(|R^+|d_l d_{l-1})$, where R^+ represents the training dataset involving the observed interactions, d_l and d_{l-1} denote the current and previous transformation size. For the prediction layer and the attention layer, the time complexity of inner product is $O(\sum_{l=1}^{L} |R^+|d_l)$. As a result, the overall time complexity of our proposed model is $O(\sum_{l=1}^{L} |R^+|d_l d_{l-1} + 2\sum_{l=1}^{L} |R^+|d_l)$.

4 Experiments

In this section, we conduct extensive experiments to evaluate our proposed algorithm. We aim to answer the following research questions:

RQ1 How does our GCNLP algorithm perform compared with other state-of-the-art recommendation algorithms?

RQ2 How do different components (LPA, different GCNs) affect the performance of GCNLP.

RQ3 How does our proposed GCNLP algorithm perform in the data sparse scenarios?

4.1 Evaluation datasets

To evaluate the effectiveness of the GCNLP algorithm, we conduct comprehensive experiments on three benchmark datasets: Movilens-1M, Last-FM, and Book-Crossing, respectively.

Movielens-1M[1]**:** This movie rating dataset has been widely used to evaluate recommendation algorithms. We employ the version containing one million ratings, where each user has at least 20 ratings.

LastFM[2]**:** This dataset contains musician listening information from a set of 2 thousand users from LastFM online music system, which provides the listening count as weight for each user-item interaction.

Book-Crossing[3]**:** This dataset contains 1 million ratings (ranging from 0 to 10) of books in the book-crossing community.

Since the above three datasets are explicit feedback, we convert them to implicit data, where each item is marked as 0 or 1, indicating whether the user interacts with the item. In addition to the user-item interactions, we also need to construct item knowledge for each dataset. In addition to the user-item interaction data, we also construct item-side KG for each dataset. Specially, we leverage Microsoft Satori to construct the knowledge graph for each dataset. We first sample a subset of triples from the entire KG. For the sub-graph, we match the dataset with triples tail. Then, we match the product ID to the beginning of all triples and sample all well-matched triples. The basic statistics of the three datasets are shown in Table 1.

[1] https://grouplens.org/datasets/movielens/1m/.
[2] https://grouplens.org/datasets/hetrec-2011/.
[3] http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

Table 1 Statistics of the three datasets

Datasets	ml-1m	LastFM	Book-crossing
Users	6036	1872	17,860
Items	2347	3846	14,910
Interactions	753,772	42,346	139,746
Entities	16,954	9366	25,787
Relations	32	60	18
KG triples	20,195	15,518	60,787

4.2 Experimental settings

4.2.1 Evaluation metrics

For each user, we randomly hold out one item that the user has interacted with (user liked), and sample 100 unobserved (user disliked) or negative items to form the test sets. The performance of the rank-list is judged by Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG). HR is a way of calculating how many "hits" a user has in an k-sized list of ranked items, and it is formulated as

$$HR@K = \frac{Number\ of\ Hits@K}{|GT|}, \tag{24}$$

where GT is the collection of all test datasets. NDCG accumulated at a particular rank position K, and it highlights the retrieval of relevant results. The metric of DCG is defined as

$$DCG@K = \sum_{i=1}^{N} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \tag{25}$$

where rel_i is the graded relevance of the recommendation result at position i. Then the $NDCG@K$ of user u is defined as

$$NDCG_u@K = \frac{DCG_u@K}{IDCG_u}, \tag{26}$$

where $IDCG_u$ represents the best recommendation list returned by a user. Then the average metric of $NDCG$ is defined as

$$NDCG@K = \frac{\sum_{u \in U} NDCG_u@K}{IDCG_u}. \tag{27}$$

Additionally, to evaluate the homogeneity of user preferences, the metric of Coverage is defined as

$$Coverage@K = \frac{|U_{u \in U} R(u)|}{|I|}, \tag{28}$$

where I represents the total item set, U is the user set, and $R(u)$ indicates that the recommender system recommends an item list of length K to the user u.

4.2.2 Baselines

We implement our GCNLP algorithm using the python library of tensorflow[4], and compare it with the following baselines:

- BPRMF [18]: This method optimizes the MF model with a pairwise ranking loss, which is tailored to learn from implicit feedback. We employ a fixed learning rate and change other parameters to report the best results.
- NeuMF [32]: It is a typical recommendation algorithm based on deep learning. It combines traditional matrix factorization and multi-layer perceptron, which can extract low-dimensional and high-dimensional features at the same time, and has a good recommendation effect.
- RippleNet [33]: This is an end-to-end framework that utilize KG for recommendation. RippleNet adopts the preference propagation in KG to continuously and automatically discover users' potential hierarchical interests.
- NGCF [7]: This method uses the bipartite neural network to encode the historical interaction information of user-item into embedding to improve the recommendation effect. More importantly, NGCF explicitly considers the high-level connectivity between user-items to further enhance the representation ability of embedding.
- LRGCN [11]: This method eliminates the nonlinearity in the GCN, thus simplifying the network structure and introduced a residual network structure to alleviate the problem of over-smoothing. Which has achieved a substantial improvement recommended accuracy on NGCF.

4.2.3 Parameter settings

In our experiments, we randomly select 70% of each dataset as training set, the rest 30% as the test set. We consider latent dimension d from $\{2, 4, 6, 8, 16, 32\}$ for all models. For baseline methods, the model parameters are set according to settings mentioned in the author's paper. For GCNLP model, the batchsize and learning rate (lr) for the three datasets are set to: $batchsize = 2048$, $lr = 0.02$ on Movielens-1M dataset; $batchsize = 128$, $lr = 0.0005$ on

[4] https://github.com//haomiaocqut/ReSys_GCNLP.

LastFM dataset; $batchsize = 256$, $lr = 0.0002$ on Book-Crossing dataset. We set the size of neighbor and the number of hops as: 8 and 4 for Movielens-1M and Book-Crossing; 4 and 2 for LastFM.

4.3 Performance comparison(RQ1)

Figure 4 shows the performance (HR@10 and NDCG@10) of all the competitive algorithms with respect to different numbers of latent dimension d. We test the values of dimension as [2, 4, 6, 8, 16, 32], respectively. With the latent dimension changes, the performance of all models fluctuates within a small range. From Fig. 4, it is easy to see that GCN-based models perform better than other models. It is worth mentioning that LRGCN is a strong baseline which beats RippleNet and NGCF on the three datasets. GCNLP and LRGCN algorithms are superior to all other algorithms on adopted three datasets. On the Book-Crossing dataset, our GCNLP algorithm is weaker to the LRGCN algorithm in HR@10 metric, but our GCNLP algorithm performs better than the LRGCN algorithm in NDCG@10 metric. In the metric of NDCG@10, the GCNLP and LRGCN algorithms are significantly outperform all other algorithms on LastFM and BookCrossing datasets. And in the metric of HR@10, GCN-based models perform significantly stronger than other models on LastFM datasets. To sum up, compared with the other baselines, our GCNLP algorithm has achieved the best experimental results on the three datasets.

Graph convolution operation is easy to cause over-smooth, and homogenizes user preferences. Figure 5 shows the performance (Coverage@10) of GCN-based models on adopted three datasets. We can find that GCNLP and LRGCN always perform best on the three datasets, which means that they perform better in alleviating the homogeneity of user preferences. The Coverage@10 of GCNLP on the three datasets are 0.362, 0.462, 0.493, respectively. Compared with the Coverage@10 of LRGCN on the three datasets are 0.354, 0.422, 0.472, our algorithm has increased by 2.9%, 9.5%, and 4.2%, respectively. To sum up, compared to other GCN-based baselines, our GCNLP algorithm always performs best on the three data sets, which fully reflects the superiority of our algorithm.

Figure 6 shows the performance of Top-K recommended lists where the ranking position K ranges from 1 to 10. It can be seen, GCNLP and LRGCN algorithms demonstrate consistent improvements over other methods across positions. We further conducted one-sample paired t-tests, verifying that all improvements are statistically significant for $p < 0.01$. For other baseline algorithms, we can see that the gap between these algorithms is very small, and their performance on the three datasets is comparable. It's worth pointing out that the NDCG value of the GCN-based model is significantly better than other models on the

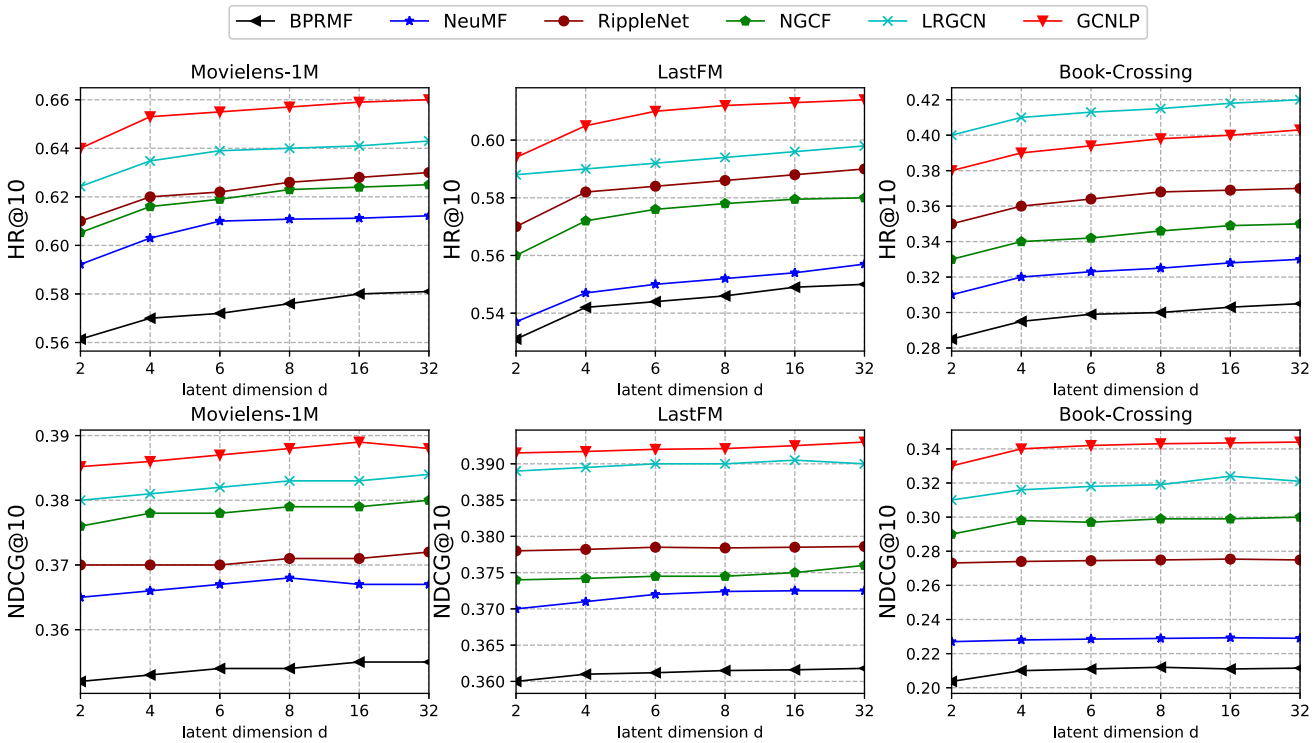


Fig. 4 The HR@10 and NDCG@10 performance of our GCNLP algorithm compared with other baselines on three datasets, with latent dimension ranging from 2 to 32

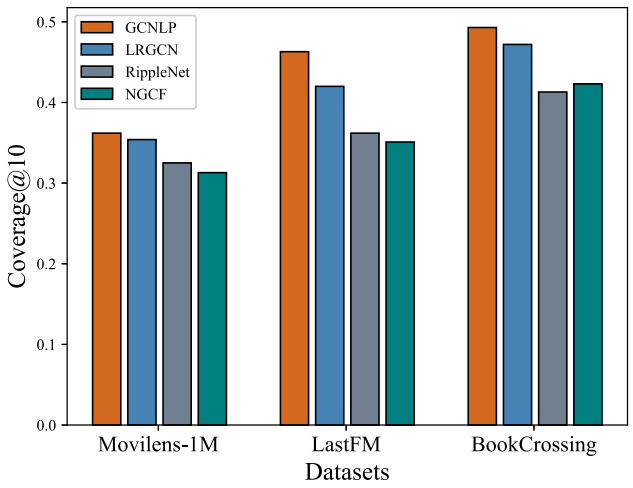


Fig. 5 Results of Coverage@10 on the three datasets

BookCrossing dataset. It can be seen that fusion of auxiliary information is very effective in improving the accuracy of the recommendation system.

In this section, we compare the GCNLP algorithm to several state-of-the-art baselines on Movielens-1M, LastFM and Book-Crossing datasets. From the experimental results of Figs. 4,5 and 6, our GCNLP algorithms typically cope better in almost all cases. In particular, the best baseline is consistently LRGCN algorithm. However, it still underperforms our GCNLP algorithm in most cases. The key reason is that the LRGCN only by simplifying GCN (i.e., reduces the number of network layers and removes nonlinear transformations) to improve performance, where the adjacency matrix A is determined by the user-item interaction matrix, thus A is always a constant. As a result, LRGCN fails to consider the importance of users to various relationships. In contrast, in the GCNLP algorithm, the (i, j)-entry $A_u{}^{ij} = c_u(r_{e_i,e_j})$ by learning edge weights to characterize the importance of users to various relationships. In particular, we leverage these embeddings as attention weights of users and items, and aggregate them to accurately capture user preferences, which shows that our algorithm combining KG and the attention mechanism is effective in improving the recommendation performance. On the other hand, our model combines the label propagation algorithm to alleviate the over smooth problem caused by GCN, so compared with the model that only uses GCN, the performance of our model is significantly improved. It is also worth mentioning that the HR@10 of LRGCN on Book-Crossing is higher than that of GCNLP, the main reason is that the types of relations we set on Book-Crossing are not enough (see Table 1), so that GCNLP's performance on Book-Crossing is limited to some extent.

4.4 Ablation studies(RQ2)

We conduct ablation studies on GCNLP by showing how do the LPA and different GCNs affect its performance. In

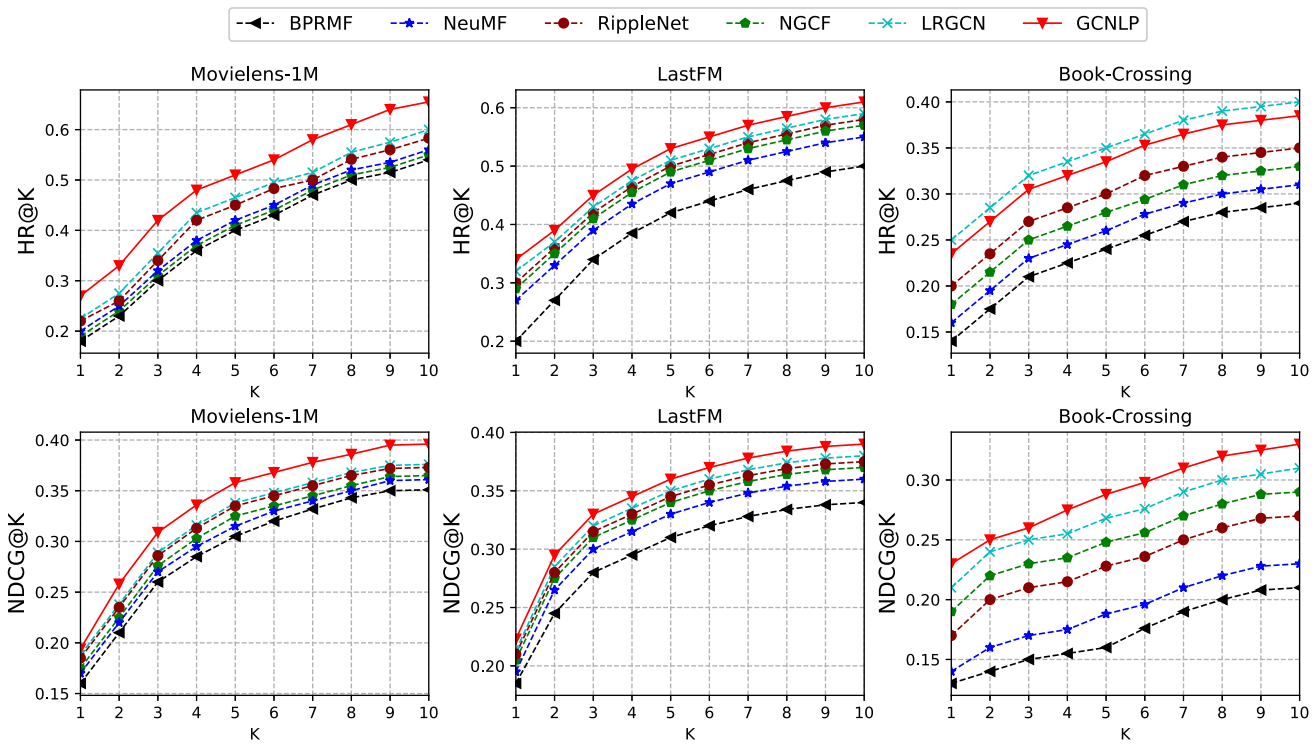


Fig. 6 Evaluation of Top-K item recommendation where K ranges from 1 to 10 on the three datasets

addition, we seek to explain why these designs are so efficient.

4.4.1 The influence of LPA on the GCNLP

In order to further study the influence of the label propagation algorithm (LPA) on GCNLP, we have done a lot of ablation experiments, removing the LPA of the model and comparing it with GCNLP.

Figure 7 shows the HR@10 performance of GCNLP compared with GCN. We can clearly see that GCNLP always performs better than GCN on the three datasets. The HR@10 of GCNLP on the three datasets are 0.660, 0.617, 0.406, respectively. Compared with the HR@10 of GCN on the three datasets are 0.622,0.582, 0.372, GCNLP has increased by 5.9%, 6.1%,and 9.1%, respectively.

Figure 8 shows the NDCG@10 performance of GCNLP compared with GCN. We have almost the same conclusion, GCNLP always performs better than GCN on the three datasets. The NDCG@10 of GCNLP on the three datasets are 0.388, 0.391, 0.340, respectively. Compared with the NDCG@10 of GCN on the three datasets are 0.355, 0.360, 0.291, GCNLP has increased by 9.2%, 8.3%,and 15.1%, respectively.

Figure 9 shows the Coverage@10 performance of GCNLP compared with GCN. As we expected, GCNLP always performs better than GCN on the three datasets. The Coverage@10 of GCNLP on the three datasets are 0.362, 0.463, 0.493 respectively. Compared with the Coeverage@10 of GCN on the three datasets are 0.293, 0.370, 0.385, GCNLP has increased by 23.5%, 24.4%,and 28.0%, respectively.

To sum up, for HR@10 and NDCG@10, the improvement of GCNLP compared with GCN is almost within 10%. Only in the BookCrossing dataset, the improvement

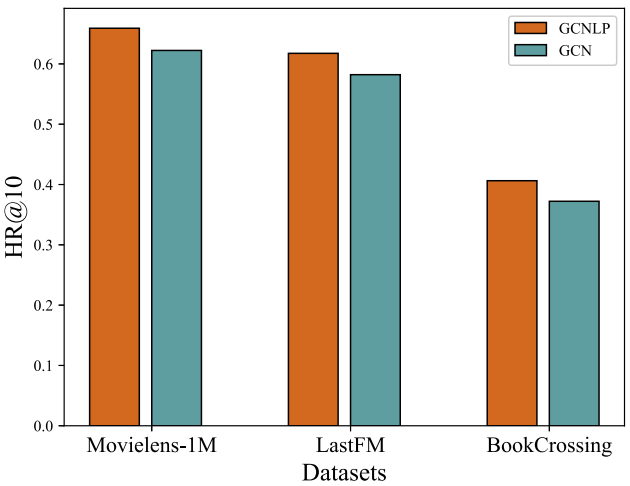


Fig. 7 The HR@10 performance of GCNLP compared with GCN on the three datasets

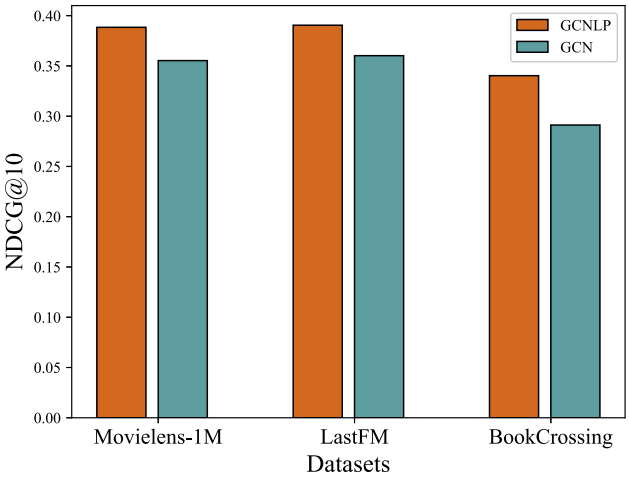


Fig. 8 The NDCG@10 performance of GCNLP compared with GCN on the three datasets

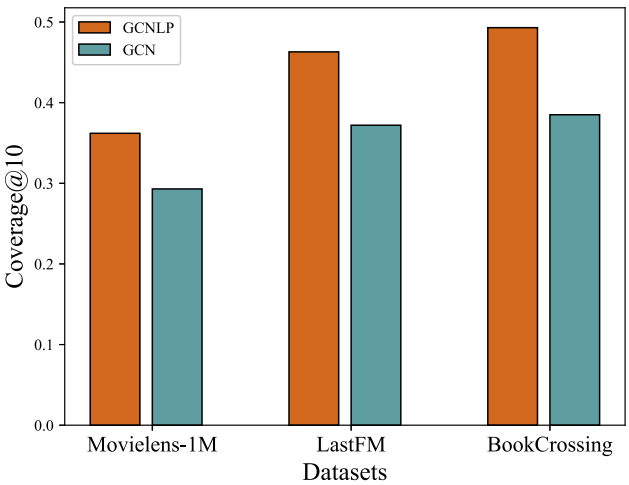


Fig. 9 The Coverage@10 performance of GCNLP compared with GCN on the three datasets

rate of NDCG@10 is 15.1%. But for Coverage@10, the improvement rate of GCNLP compared with GCN exceeds 20% on the three datasets. This means that LPA not only has a certain degree of positive impact on improving the accuracy of the model, but also has a significant increase in the coverage rate of the model, and can significantly alleviate the over smoothing caused by GCN.

4.4.2 The influence of different GCNs on recommender model

To investigate the influence of different GCNs on recommender model, we also conduct experiments combining traditional GCN and LPA. It is worth noting that in the traditional GCN-based method, the adjacency matrix A is directly determined by the user-item interaction matrix Y, which as shown in Eq. (29), and the rest parts of the model

Table 2 The comparison of performance (HR@20 and NDCG@20) between T-GCNLP and GCNLP

Datasets	Movielens-1M		LastFM		Book-crossing	
Methods	HR	NDCG	HR	NDCG	HR	NDCG
T-GCNLP	0.753	0.402	0.652	0.381	0.542	0.356
GCNLP	**0.787**	**0.437**	**0.712**	**0.421**	**0.586**	**0.384**
Improv.	3.9%	7.5%	9.2%	10.4%	8.1%	8.4%

Bold values represent the largest value

remain the same. We abbreviate the combination model as T-GCNLP. For fair comparison, we fix all hyper-parameters (e.g., size of embedding, learning rate and batchsize, etc.). And we report the results of HR@20 and NDCG@20 on the three benchmark datasets in Table 2.

$$A = \begin{pmatrix} 0 & Y \\ Y^T & 0 \end{pmatrix} \tag{29}$$

It is shown in Table 2 that compared to T-GCNLP, our model has a significant improvement. Intuitively, this is due to the fact that our GCN is superior to traditional GCN. The key reason is that in our model, the (i, j)-entry $A_u{}^{ij} = c_u(r_{e_i,e_j})$ by learning edge weights to characterize the importance of users to various relationships. However, in traditional GCN, A is always a constant, the relations between entities are not explicitly modeled, and fails to consider the fact that users attach different degrees of importance to various relations of items. As a result, GCNLP performs more effective than T-GCNLP.

4.5 Sparse scenarios analysis (RQ3)

Table 3 shows the experimental results with different training set (10% to 90%) ratios on Movielens-1M, the rest datasets as test set, while keeping other parameters fixed. We omit the results on LastFM and Book-Crossing because we reached the same conclusion: Integrating KG into recommender models can alleviate the data sparsity to a certain extent. From Table3, we observe that the performance of all algorithms deteriorates with the reduce of the training set. When r=10%, HR@20 decreases by 7.6%, 7.2%, 4.5%, 4.2%, 4.4%, 5.2% for BPRMF, NeuMF, RippleNet, NGCF, LRGCN, and GCNLP, respectively, compared with the case of$r$=90%. This experimental result shows that even in the case of sparse user-item interaction, the performance of the GCN-based model is far superior to other algorithms. We also notice that the GCNLP algorithm is more sensitive to the density of user-item interactions, but in general, it performs better than the LRGCN algorithm in data sparse scenarios.

Overall, the data sparsity is a critical challenge in the recommendation scenario. If we solely utilize user ratings as input data, it is tough to use more information to recommend items that satisfy users in data sparse scenarios. Leveraging KG as auxiliary information is a effective method to improve performance in data sparse scenarios. The KG is useful solution to address this issue due to it provides rich semantic information. By comparing the experimental results of our GCNLP algorithm under the data sparse scenarios, it is shown that the GCNLP algorithm is effective in addressing the data sparsity to large extent.

5 Conclusion

In this work, we developed a recommender model unifying graph convolutional networks and label propagation algorithms. Our key argument is that GCN-based methods benefit from both the semantic embedding of KG and the semantic path pattern, which naturally fit the process of embedding propagation. However, existing GCN-based methods are easy to cause over-smoothing, and make the preference of different users become homogeneous. To address this problem, we utilize the label propagation algorithm to assist the GCN model to regularize edge weights. In addition, we introduce an attention network to capture the attention weight of each user-item pair, which takes into account the fact that users attach different degrees of importance to the various relationships of items. We conduct comprehensive experiments on three public

Table 3 Results of HR@20 on MovieLens-1M with different ratios of training set

Model	Ratios of training set									%Change
	10%	20%	30%	40%	50%	60%	70%	80%	90%	
BPRMF	0.6523	0.6620	0.6746	0.6688	0.6803	0.6838	0.6983	0.7097	0.7033	7.6
NeuMF	0.6811	0.6869	0.6942	0.7065	0.7078	0.7110	0.7292	0.7385	0.7345	7.2
RippleNet	0.7230	0.7218	0.7305	0.7360	0.7445	0.7472	0.7454	0.7551	0.7518	**4.5**
NGCF	0.7122	0.7376	0.7380	0.7357	0.7404	0.7471	0.7488	0.7439	0.7441	**4.2**
LRGCN	0.7470	0.7548	0.7537	0.7646	0.7748	0.7729	0.7741	0.7816	0.7801	**4.4**
GCNLP	**0.7605**	**0.7709**	**0.7758**	**0.7742**	**0.7797**	**0.7823**	**0.7871**	**0.7910**	**0.7995**	**5.2**

Bold values represent the largest value

datasets to demonstrate the effectiveness of our GCNLP algorithm. Experimental results show that our algorithm outperforms state-of-the-art algorithms.

As future work, we are particularly interested in exploring dynamic recommendation. Although the GCN-based recommendation model has achieved prominent progress due to the remarkable ability on representation learning, the training process is still time-consuming. In many circumstances,, such as online shopping, news recommendations, and forums, users' preferences may be affected by social activities in a short period of time [34]. In this case, recommendations using static preference modeling may not capture the user's current interest. An effective way to solve this problem is to explore group interests and exploit the connections between group users to collect a comparably compact candidate set of potential media-user pairs [35]. Another solution is to leverage a dynamic graph network to capture users' ever-changing preferences by combining long term and short term interests of friends [36]. As a result, we deem that integrating other sorts of auxiliary information and from the KG for dynamic recommendation is the future trend.

Acknowledgements The work is supported by the National Natural Science Foundation of China (No. 61702063), the Natural Science Foundation of Chongqing (No. cstc2019jcyj-msxmX0544), the Science and Technology Research Program of Chongqing Municipal Education Commission (Nos. KJZD-K202101105, KJQN202001136).

Declarations

Conflict of interest We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work. There is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled, "Integrating Label Propagation with Graph Convolutional Networks for Recommendation".

References

1. Nápoles G, Grau I, Salgueiro Y (2020) Recommender system using long-term cognitive networks. Knowl-Based Syst 206:106372
2. Da'u A, Salim N (2020) Recommendation system based on deep learning methods: a systematic review and new directions. Artif Intell Rev 53(4):2709–2748
3. Rendle S, Gantner Z, Freudenthaler C, Schmidt-Thieme L (2011) Fast context-aware recommendations with factorization machines. In: Proceedings of the 34th international ACM SIGIR conference on research and development in information retrieval, pp 635–644
4. He X, Chua T-S (2017) Neural factorization machines for sparse predictive analytics. In: Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval, pp 355–364
5. Zhang L, Shen W, Huang J, Li S, Pan G (2019) Field-aware neural factorization machine for click-through rate prediction. IEEE Access 7:75032–75040
6. Wang X, He X, Cao Y, Liu M, Chua T-S (2019) Kgat: knowledge graph attention network for recommendation. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 950–958
7. Wang X, He X, Wang M, Feng F, Chua T-S (2019) Neural graph collaborative filtering. In: Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval, pp 165–174
8. Liu F, Cheng Z, Zhu L, Liu C, Nie L (2020) A2-gcn: an attribute-aware attentive gcn model for recommendation. IEEE Transactions on Knowledge and Data Engineering, p 1
9. Liu F, Cheng Z, Zhu L, Gao Z, Nie L (2021) Interest-aware message-passing gcn for recommendation. In: Proceedings of the web conference 2021:1296–1305
10. He X, Deng K, Wang X, Li Y, Zhang Y, Wang M (2020) Lightgcn: simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, pp 639–648
11. Chen L, Wu L, Hong R, Zhang K, Wang M (2020) Revisiting graph based collaborative filtering: a linear residual graph convolutional network approach. In: Proceedings of the AAAI conference on artificial intelligence 34(01):27–34
12. Wu F, Souza A, Zhang T, Fifty C, Yu T, Weinberger K (2019) Simplifying graph convolutional networks. In: International conference on machine learning. PMLR, pp 6861–6871
13. Wang H, Leskovec J (2020) Unifying graph convolutional neural networks and label propagation. arXiv preprint arXiv:2002.06755
14. Huang Q, He H, Singh A, Lim S-N, Benson AR (2020) Combining label propagation and simple models out-performs graph neural networks. arXiv preprint arXiv:2010.13993
15. Cai H, Zheng VW, Chang KC-C (2018) A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans Knowl Data Eng 30(9):1616–1637
16. Zhang F, Yuan NJ, Lian D, Xie X, Ma W-Y (2016) Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 353–362
17. Bordes A, Usunier N, Garcia-Duran A, Weston J, Yakhnenko O (2013) Translating embeddings for modeling multi-relational data. In: Neural information processing systems (NIPS), pp 1–9
18. Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L (2012) Bpr: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618
19. Wang H, Zhang F, Xie X, Guo M (2018) Dkn: deep knowledge-aware network for news recommendation. In: Proceedings of the 2018 world wide web conference, pp 1835–1844
20. Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the conference on empirical methods in natural language processing (EMNLP)
21. Wang H, Zhang F, Hou M, Xie X, Guo M, Liu Q (2018) Shine: signed heterogeneous information network embedding for sentiment link prediction. In: Proceedings of the eleventh ACM international conference on web search and data mining, pp 592–600
22. Cao Y, Wang X, He X, Hu Z, Chua T-S (2019) Unifying knowledge graph learning and recommendation: towards a better understanding of user preferences. In: The world wide web conference, pp 151–161
23. Yu X, Ren X, Gu Q, Sun Y, Han J (2013) Collaborative filtering with entity similarity regularization in heterogeneous information networks. IJCAI HINA, vol 27

24. Luo C, Pang W, Wang Z, Lin C (2014) Hete-cf: social-based collaborative filtering recommendation using heterogeneous relations. In: 2014 IEEE international conference on data mining. IEEE, pp 917–922
25. Yu X, Ren X, Sun Y, Sturt B, Khandelwal U, Gu Q, Norick B, Han J (2013) Recommendation in heterogeneous information networks with implicit user feedback. In: Proceedings of the 7th ACM conference on recommender systems, pp 347–350
26. Yu X, Ren X, Sun Y, Gu Q, Sturt B, Khandelwal U, Norick B, Han J (2014) Personalized entity recommendation: a heterogeneous information network approach. In: Proceedings of the 7th ACM international conference on Web search and data mining, pp 283–292
27. Zhao H, Yao Q, Li J, Song Y, Lee DL (2017) Meta-graph based recommendation fusion over heterogeneous information networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 635–644
28. Ma W, Zhang M, Cao Y, Jin W, Wang C, Liu Y, Ma S, Ren X (2019) Jointly learning explainable rules for recommendation with knowledge graph. In: The world wide web conference, pp 1210–1221
29. Wang Z, Lin G, Tan H, Chen Q, Liu X (2020) Ckan: collaborative knowledge-aware attentive network for recommender systems. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp 219–228
30. Jin J, Qin J, Fang Y, Du K, Zhang W, Yu Y, Zhang Z, Smola AJ (2020) "An efficient neighborhood-based interaction model for recommendation on heterogeneous graph. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp 75–84
31. Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K-i, Jegelka S (2018) Representation learning on graphs with jumping knowledge networks. In: International conference on machine learning. PMLR, pp 5453–5462
32. He X, Liao L, Zhang H, Nie L, Hu X, Chua T-S (2017) Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web, pp 173–182
33. Wang H, Zhang F, Wang J, Zhao M, Li W, Xie X, Guo M (2018) Ripplenet: propagating user preferences on the knowledge graph for recommender systems. In: Proceedings of the 27th ACM international conference on information and knowledge management, pp 417–426
34. Guo Q, Zhuang F, Qin C, Zhu H, Xie X, Xiong H et al (2020) A survey on knowledge graph-based recommender systems. Sci Sinica Inf 50(7):937
35. Qin D, Zhou X, Chen L, Huang G, Zhang Y (2018) Dynamic connection-based social group recommendation. IEEE Trans Knowl Data Eng 32(3):453–467
36. Song W, Xiao Z, Wang Y, Charlin L, Zhang M, Tang J (2019) Session-based social recommendation via dynamic graph attention networks. In: Proceedings of the twelfth ACM international conference on web search and data mining, 2019, pp 555–563