Effective Knowledge-Aware Recommendation via Graph Convolutional Networks

Bo Zhao, Zhuoming Xu(✉), Yan Tang, Jian Li, Bei Liu, and Haimei Tian

College of Computer and Information, Hohai University, Nanjing 210098, China
{bzhao, zmxu, tangyan, jli, liubei, hmtian}@hhu.edu.cn

Abstract. Most existing graph neural network (GNN)-based knowledge-aware recommendation models rely on handcrafted feature engineering and do not allow for end-to-end training. As a state-of-the-art end-to-end framework, the Knowledge-aware Graph Neural Networks with Label Smoothness Regularization (KGNN-LS) model can extend GNNs architecture to knowledge graphs to simultaneously capture semantic relations between entities as well as personalized user preferences for entities/items, thereby making effective recommendation. However, we believe that KGNN-LS still has two weaknesses: (1) In KGNN-LS, the weights of the edges in the graph are determined solely by user preferences for relations without considering user's (potential) personalized interests in entities/items. (2) The sum pooling adopted by KGNN-LS cannot effectively aggregate the most representative information of the neighborhood. In this paper, we propose the improved Knowledge-aware Graph Neural Networks with Label Smoothness Regularization (iKGNN-LS) model, which makes two improvements to KGNN-LS: (1) In iKGNN-LS, by introducing user-specific entity scoring functions, the edge weights are determined jointly by personalized user preferences for relations and for entities. (2) iKGNN-LS uses max pooling instead of sum pooling for neighborhood aggregation. Top-N recommendation experiments on three datasets show that iKGNN-LS outperforms KGNN-LS in terms of Precision@N, Recall@N, and F1-measure@N.

Keywords: Knowledge-aware recommendation · Knowledge graph · GCN · User-specific entity scoring function · Pooling aggregator

1 Introduction

Knowledge graphs (KGs) [5] have proven to be effective in enhancing recommendation performance by providing recommender systems with additional knowledge [2, 4, 7–10, 12, 13]. KG-based recommender systems exploit knowledge-aware recommendation models and apply KGs in three ways [3]: embedding-based methods, path-based methods, and unified methods. *Embedding-based methods* [2, 13] use KG embedding algorithms to translate KG elements into low-dimensional vector representations, which are further integrated into the recommendation models. *Path-based methods* [4, 10] leverage the informative connectivity patterns between the entities in the user-item KG for recommendation. *Unified methods* [7–9] leverage both the semantic

G. Wang et al. (Eds.): WISA 2020, LNCS 12432, pp. 96–107, 2020.
https://doi.org/10.1007/978-3-030-60029-7_9

representation and the connectivity information in the KG for recommendation. Unified methods generally adopt an architecture based on graph neural networks (GNNs) [11], such as graph convolutional networks (GCNs) [6].

However, most existing GNN-based knowledge-aware recommendation models rely on handcrafted feature engineering and do not allow for end-to-end training [8]. The Knowledge-aware Graph Neural Networks with Label Smoothness Regularization (KGNN-LS) model recently proposed by H. Wang *et al.* [7, 8] is a state-of-the-art end-to-end framework that can extend GNNs architecture to KGs to simultaneously capture semantic relations between entities as well as personalized user preferences for entities/items, thereby making effective recommendation.

Despite a state-of-the-art model, KGNN-LS still has two weaknesses: (1) In KGNN-LS, the weights of the edges in the user-specific weighted graph are determined solely by user preferences for relations without considering user's personalized interests in entities/items. This approach cannot distinguish the different contributions of different neighbor entities connected by the same relation to user interests. (2) The sum pooling adopted by KGNN-LS cannot effectively aggregate the most representative information of the neighborhood, resulting in less effective user-specific item embeddings.

To overcome the weaknesses of KGNN-LS, in this paper we propose the improved Knowledge-aware Graph Neural Networks with Label Smoothness Regularization (iKGNN-LS) model, which makes two improvements to KGNN-LS: (1) In iKGNN-LS, by introducing user-specific entity scoring functions, the user-specific edge weights are determined jointly by personalized user preferences for relations and for entities. (2) During feature propagation on the user-specific weighted graph, iKGNN-LS uses max pooling instead of sum pooling for neighborhood aggregation.

To verify the effectiveness of iKGNN-LS and its performance advantage over KGNN-LS, we conducted two comparative experiments of top-N recommendation. First, we used the MovieLens 20M dataset and KGNN-LS to study the influences of edge weights and pooling aggregators on the performance of top-N recommendation. The results show that the edge weights determined jointly by user preferences for relations and for entities, as well as the use of max pooling in neighborhood aggregation can improve recommendation performance. Second, we used three datasets, MovieLens 20M, Last.FM, and Yelp2018, to compare the top-N recommendation performance between KGNN-LS and iKGNN-LS. The results indicate that iKGNN-LS outperforms KGNN-LS in terms of Precision@N, Recall@N, and F1-measure@N.

In summary, the main contributions of this paper are as follows:

- We propose the knowledge-aware recommendation model iKGNN-LS that can calculate user-specific item embeddings more effectively. The model exploits user preferences for relations and for entities to jointly determine user-specific edge weights and uses max pooling instead of sum pooling to aggregate the most representative information of the neighborhood.
- Our comparative experiment on KGNN-LS indicates that both the improved edge weights and max pooling can improve recommendation performance.
- Our comparative experiment between KGNN-LS and iKGNN-LS demonstrates the recommendation performance advantage of iKGNN-LS over KGNN-LS.

2 Improved Knowledge-Aware Recommendation Model

In this section, we expatiate on our proposed iKGNN-LS model. We first give the overview of the model, and then describe three major steps of the model in detail.

In the following, we first introduce some concepts and notations, and then formulate the problem of knowledge-aware recommendation.

A recommender system (RS) has a set of m users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and a set of n items $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$. According to users' implicit feedback, a user-item interaction matrix $\mathbf{Y} \in \mathbb{R}^{m \times n}$ can be defined for the system. The matrix's element $y_{uv} = 1$ indicates that user $u \in \mathcal{U}$ engages with item $v \in \mathcal{V}$, such as clicking, browsing, or purchasing; otherwise $y_{uv} = 0$.

A knowledge graph $\mathcal{G}(\mathcal{E}, \mathcal{R})$ has a set of entities $\mathcal{E}$ and a set of relations $\mathcal{R}$. The graph consists of entity-relation-entity triples (h, r, t), where $h \in \mathcal{E}$, $r \in \mathcal{R}$, and $t \in \mathcal{E}$ are the head, relation, and tail of a triple. In the recommendation setting, each item $v \in \mathcal{V}$ in the RS corresponds to an entity $e \in \mathcal{E}$ in the knowledge graph.

As formulated in [7, 8], given the interaction matrix $\mathbf{Y}$ and the knowledge graph $\mathcal{G}$, the goal of knowledge-aware recommendation is to predict whether user $u \in \mathcal{U}$ has potential interest in item $v \in \mathcal{V}$ with which the user has not engaged before. That is, the task is to learn a prediction function $\tilde{y}_{uv} = \mathcal{F}(u, v|\Theta, \mathbf{Y}, \mathcal{G})$, where $\tilde{y}_{uv}$ denotes the probability that user u will engage with item v, and Θ are model parameters of function $\mathcal{F}$.

2.1 Overview

The main goal of our proposed iKGNN-LS is to learn user-specific item embeddings more effectively. The overview of iKGNN-LS is depicted in Fig. 1, where the learning and recommendation process can be divided into the following steps:

- *Transforming the KG into a user-specific weighted graph:* The KG is transformed into a user-specific weighted graph, where a limited number of direct neighbors of each entity are sampled, and each relation (edge) between two sampled entities is given a user-specific weight to reflect the user's personalized interest.
- *Learning the prediction function:* iKGNN-LS takes the user-specific weighted graph as input, and uses knowledge-aware GCN to calculate user-specific item embeddings via feature propagation on the graph. Simultaneously, it performs label smoothness regularization on the edge weights via label propagation on the graph. Finally, iKGNN-LS uses the unified loss function to learn the prediction model.
- *Making top-N recommendation:* iKGNN-LS uses the learned prediction function to predict probabilities that the user will engage with candidate items, and then employs the probabilities to produce a top-N recommendation list for the user.

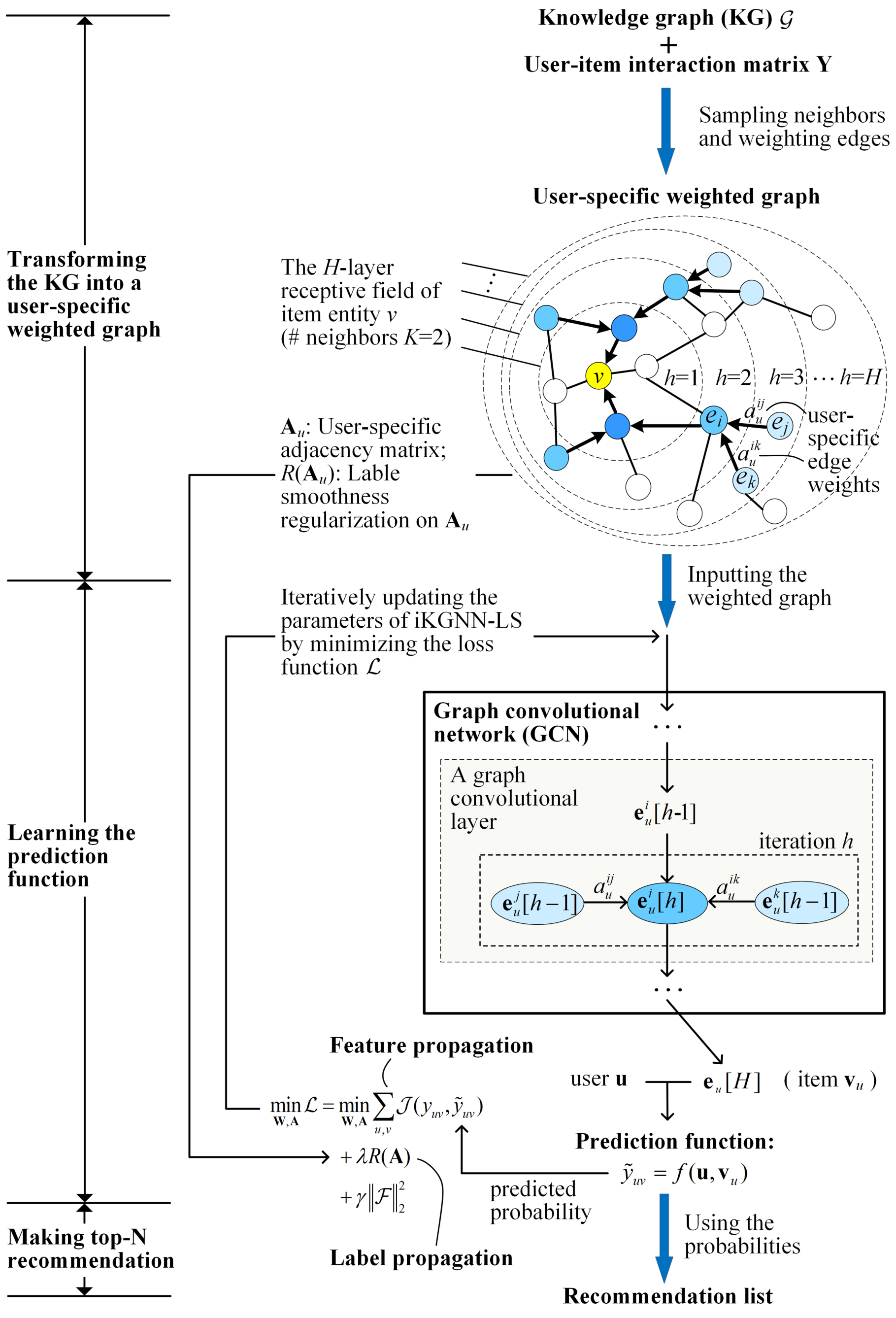
Knowledge graph (KG) $\mathcal{G}$
+
User-item interaction matrix $\mathbf{Y}$
Sampling neighbors and weighting edges
User-specific weighted graph
Transforming the KG into a user-specific weighted graph
The H-layer receptive field of item entity v (# neighbors K=2)
v
h=1 h=2 h=3 $\cdots$ h=H
e_i e_j e_k
a_u^{ij} a_u^{ik}
user-specific edge weights
$\mathbf{A}_u$: User-specific adjacency matrix; $R(\mathbf{A}_u)$: Lable smoothness regularization on $\mathbf{A}_u$
Inputting the weighted graph
Iteratively updating the parameters of iKGNN-LS by minimizing the loss function $\mathcal{L}$
Learning the prediction function
Graph convolutional network (GCN)
A graph convolutional layer
$\mathbf{e}_u^i[h\text{-}1]$
iteration h
$\mathbf{e}_u^j[h-1]$ $\mathbf{e}_u^i[h]$ $\mathbf{e}_u^k[h-1]$
Feature propagation
$\min_{\mathbf{W},\mathbf{A}} \mathcal{L} = \min_{\mathbf{W},\mathbf{A}} \sum_{u,v} \mathcal{J}(y_{uv}, \tilde{y}_{uv})$
$+\lambda R(\mathbf{A})$
$+\gamma \|\mathcal{F}\|_2^2$
Label propagation
user $\mathbf{u}$ $\mathbf{e}_u[H]$ (item $\mathbf{v}_u$)
Prediction function: $\tilde{y}_{uv} = f(\mathbf{u}, \mathbf{v}_u)$
predicted probability
Using the probabilities
Recommendation list
Making top-N recommendation

Fig. 1. Overview of our proposed iKGNN-LS model

2.2 Transforming the KG into a User-Specific Weighted Graph

Given a user in the RS, iKGNN-LS transforms the KG into a user-specific weighted graph [8]. In such a graph, at most K direct neighbors of each entity node are sampled (concerned), an edge (representing a relation) between each sampled entity pair is given a user-specific weight to reflect the user's personalized interest in the relation.

Let $N(e_i)$ represent a set of neighbor entities directly connected to entity $e_i \in \mathcal{E}$ in $\mathcal{G}(\mathcal{E}, \mathcal{R})$. In a real-world KG, the number of entities in $N(e_i)$ may vary greatly. Using the same approach as in KGNN-LS [7, 8], iKGNN-LS samples at most K direct neighbors of each entity to form the (single-layer) receptive field $S(e_i) \triangleq \{e_j | e_j \sim N(e_i)\}$, $|S(e_i)| = K$ of entity e_i. This receptive field is used to compute the user-specific neighborhood representation of e_i, denoted as $\mathbf{e}_u^{S(e_i)}$, where $u \in \mathcal{U}$ is a specific user. $\mathbf{e}_u^{S(e_i)}$ can capture structural proximity among entities in the KG. As stated in [7], the receptive field can be extended to multiple hops away (i.e., multiple layers) to model high-order structural proximity and capture users' potential interests. As in KGNN-LS, iKGNN-LS uses the h-layer receptive field to compute the h-order structural proximity, denoted as $\mathbf{e}_u^{S(e_i)}[h],\ h = 1, 2, \ldots, H$, where H is the maximum depth of the receptive field. Figure 1 depicts the H-layer receptive field of item entity $v \in \mathcal{V} \subseteq \mathcal{E}$, where $K = 2$.

In each layer of receptive field, user-specific edge weights are calculated as follows. Given two entities $e_i \in \mathcal{E}$, $e_j \in S(e_i)$ and their relation r_{e_i,e_j}, iKGNN-LS uses user-specific relation scoring function $s_u(r_{e_i,e_j})$ defined as Eq. (1) [7, 8] to calculate the user-relation score between $u \in \mathcal{U}$ and r_{e_i,e_j}, and uses user-specific entity scoring function $t_u(e_j)$ defined as Eq. (2) to calculate the user-entity score between u and e_j.

$$s_u(r_{e_i,e_j}) = g(\mathbf{u}, \mathbf{r}_{e_i,e_j}) \tag{1}$$

$$t_u(e_j) = g(\mathbf{u}, \mathbf{e}_u^j) \tag{2}$$

where $\mathbf{u},\ \mathbf{r}_{e_i,e_j},\ \mathbf{e}_u^j \in \mathbb{R}^d$ are the representations of u, r_{e_i,e_j}, and e_j, respectively. d is the dimension of representations. g is a differentiable function (e.g., inner product).

The above two scores are normalized separately, that is, the user-relation score is normalized to $\tilde{s}_u(r_{e_i,e_j})$, as defined by Eq. (3) [7], and the user-entity score is normalized to $\tilde{t}_u(e_j)$, as defined by Eq. (4).

$$\tilde{s}_u(r_{e_i,e_j}) = \frac{\exp(s_u(r_{e_i,e_j}))}{\sum_{e \in S(e_i)} \exp(s_u(r_{e_i,e}))} \tag{3}$$

$$\tilde{t}_u(e_j) = \frac{\exp(t_u(e_j))}{\sum_{e \in S(e_i)} \exp(t_u(e))} \tag{4}$$

iKGNN-LS uses Eq. (5) to compute the weight of the edge (representing relation r_{e_i,e_j}) with respect to user u, which is referred to as user-specific edge weight.

$$a_u^{ij} = \tilde{s}_u(r_{e_i,e_j}) \cdot \tilde{t}_u(e_j) \tag{5}$$

The weight a_u^{ij} is used as an element to form a user-specific adjacency matrix $\mathbf{A}_u \in \mathbb{R}^{\mathcal{E}\times\mathcal{E}}$, which represents user-specific edge weights for the weighted graph.

2.3 Learning the Prediction Function

As mentioned earlier, our task is to learn a prediction function $\mathcal{F}(u, v|\Theta, \mathbf{Y}, \mathcal{G})$. The learning process includes three steps: feature propagation on the graph, label propagation on the graph, and model learning. Below we describe these three steps.

Feature Propagation on the Graph. The goal of feature propagation is to calculate user-specific item embeddings. The calculation process takes the user-specific weighted graph (including the H-layer receptive field and user-specific edge weights) as input, and employs GCN to compute the final H-order entity representation by aggregating and incorporating neighborhood information (i.e., structure information) in an iterative layer-by-layer manner. The total number of neighborhood aggregation iterations is H. In h-th iteration ($h = 1, 2, \ldots, H$), iKGNN-LS uses max-pooling instead of sum-pooling to aggregate all entities in $S(e_i)$ to form the $(h-1)$-order neighborhood representation of entity e_i, denoted as $\mathbf{e}_u^{S(e_i)}[h-1]$, which is defined by Eq. (6).

$$\mathbf{e}_u^{S(e_i)}[h-1] = \max(\{a_u^{ij} \times \mathbf{e}_u^j[h-1],\ \forall e_j \in S(e_i)\}) \tag{6}$$

where a_u^{ij} is the user u specific weight of the edge between e_i and e_j, and $\mathbf{e}_u^j[h-1]$ denotes the $(h-1)$-order representation of entity $e_j \in S(e_i)$.

Like KGNN-LS, iKGNN-LS then combines neighborhood representation $\mathbf{e}_u^{S(e_i)}[h-1]$ with the $(h-1)$-order representation of the entity itself, $\mathbf{e}_u^i[h-1]$, to form its h-order entity representation $\mathbf{e}_u^i[h]$, which is defined as Eq. (7) [7].

$$\mathbf{e}_u^i[h] = \sigma(\mathbf{W}_h \cdot (\mathbf{e}_u^i[h-1] + \mathbf{e}_u^{S(e_i)}[h-1]) + \mathbf{b}_h) \tag{7}$$

where $\mathbf{W}_h$ and $\mathbf{b}_h$ are transformation weight and bias, and σ is the nonlinear function such as *ReLU*.

For item entity $v \in \mathcal{V} \subseteq \mathcal{E}$, after H iterations, the final H-order entity representation $\mathbf{e}_u[H]$ (i.e., $\mathbf{v}_u[H]$) is the user-specific item embedding $\mathbf{v}_u$. We can thus input $\mathbf{v}_u$ and user representation $\mathbf{u}$ into a differentiable function (e.g., inner product) to predict the probability $\tilde{y}_{uv}$ that user u will engage with item v, as defined by Eq. (8) [7, 8].

$$\tilde{y}_{uv} = f(\mathbf{u}, \mathbf{v}_u) \tag{8}$$

Label Propagation on the Graph. The goal of label propagation is to assist the learning of entity representations and to help predict unobserved user-item interactions through label smoothness (LS) regularization on the edge weights [8]. As formulated in [8], let $l_u : \mathcal{E} \rightarrow \mathbb{R}$ denote a real-valued label function on $\mathcal{G}$, which is constrained to take a specific value $l_u(v) = y_{uv} \in \mathbf{Y}$ at node $v \in \mathcal{V} \subseteq \mathcal{E}$. The label smoothness assumption that adjacent entities in the graph are likely to have similar relevancy labels [8] leads to the following definition of energy function E.

$$E(l_u, \mathbf{A}_u) = \frac{1}{2} \sum_{e_i \in \mathcal{E}, e_j \in \mathcal{E}} a_u^{ij} (l_u(e_i) - l_u(e_j))^2 \tag{9}$$

where a_u^{ij} is the user specific edge weight. $l_u(e_i)$ and $l_u(e_j)$ are user relevancy scores of e_i and e_j, respectively. Like the approach in KGNN-LS, iKGNN-LS repeats the following two steps [8] to achieve the minimum-energy of function E, thereby predicting a user relevancy label/score for each unlabeled entity:

- Propagate labels for all entities: $l_u(\mathcal{E}) \leftarrow \mathbf{D}_u^{-1} \mathbf{A}_u l_u(\mathcal{E})$, where $l_u(\mathcal{E})$ is the vector of labels for all entities, $\mathbf{D}_u$ is a diagonal degree matrix with $D_u^{ij} = \sum_j a_u^{ij}$.
- Reset labels of all items to initial labels: $l_u(\mathcal{V}) \leftarrow \mathbf{Y}[\,u, \mathcal{V}\,]^\top$, where $l_u(\mathcal{V})$ is the vector of labels for all items and $\mathbf{Y}[\,u, \mathcal{V}\,] = [y_{uv_1}, y_{uv_2}, \ldots]$ are initial labels.

iKGNN-LS uses the same approach as KGNN-LS to perform label smoothness (LS) regularization on the edge weights. Specifically, as described in [8], a single item $v \in \mathcal{V} \subseteq \mathcal{E}$ is held out and it is treated as unlabeled; the label of v can then be predicted by using the rest of (labeled) items and (unlabeled) non-item entities. The LS regularization on the edge weights can thus be achieved via a learning procedure that uses the difference between the true relevancy label of v (i.e., y_{uv}) and the predicted label $\tilde{l}_u(v)$ as a supervised signal. The regularization is defined as Eq. (10) [8].

$$R(\mathbf{A}) = \sum_u R(\mathbf{A}_u) = \sum_u \sum_v \mathcal{J}(y_{uv}, \tilde{l}_u(v)) \tag{10}$$

where $\mathcal{J}$ is the cross-entropy loss function.

Model Learning via the Unified Loss Function. iKGNN-LS uses the same loss function as iKGNN-LS to learn the prediction model. The unified loss function, which combines knowledge-aware GCN and LS regularization, is defined as Eq. (11) [8].

$$\min_{\mathbf{W},\mathbf{A}} \mathcal{L} = \min_{\mathbf{W},\mathbf{A}} \sum_{u,v} \mathcal{J}(y_{uv}, \tilde{y}_{uv}) + \lambda R(\mathbf{A}) + \gamma \|\mathcal{F}\|_2^2 \tag{11}$$

where $\mathcal{J}$ is the cross-entropy loss function, λ and γ are balancing hyper-parameters, $R(\mathbf{A})$ is LS regularization on edge weights $\mathbf{A}$, $\|\mathcal{F}\|_2^2$ is the L2-regularizer. By minimizing the loss function, iKGNN-LS uses stochastic gradient descent (SGD) to simultaneously update model parameters: transformation matrix $\mathbf{W}$ and edge weights $\mathbf{A}$.

Note that in Eq. (11), the first term corresponds to feature propagation on the KG, whereas the second term $R(\mathbf{A})$ corresponds to label propagation on the KG. Once the trainable parameters are learned, the prediction function of iKGNN-LS is achieved.

2.4 Making Top-N Recommendation

For a specific user $u \in \mathcal{U}$ in the RS, iKGNN-LS can use the learned prediction function to compute a predicted probability that user u will engage with item $v \in \mathcal{V} \subseteq \mathcal{E}$ with which the user has not engaged before. As shown in Fig. 1, given user representation $\mathbf{u}$ and user-specific item embedding $\mathbf{v}_u$, prediction function $\tilde{y}_{uv} = f(\mathbf{u}, \mathbf{v}_u)$ (being inner product $< \mathbf{u}, \mathbf{v}_u >$ in our experiments) generates predicted probability $\tilde{y}_{uv}$. iKGNN-LS sorts the probabilities in descending order to produce a top-N recommendation list for the user.

3 Experiments

This section presents our two parts of top-N recommendation experiments: (1) we used KGNN-LS to study the influences of edge weights and pooling aggregators on top-N recommendation performance; (2) we used three datasets to compare the top-N recommendation performance of KGNN-LS and iKGNN-LS in order to show the performance advantages of iKGNN-LS over KGNN-LS. It is worth noting that the experiments in [8] have shown that KGNN-LS outperforms six state-of-the-art baselines. Therefore, our experiments do not need to compare iKGNN-LS with the baselines.

3.1 Experimental Setup

Datasets. Our experiments used the MovieLens 20M, Last.FM, and Yelp2018 datasets for movie, music, and local business recommendations. The first two datasets were published by H. Wang *et al.* [7, 8] on GitHub[1]. The authors used Microsoft Satori to construct the KGs for the MovieLens 20M and Last.FM datasets. The details on the datasets and the corresponding KGs can be found in [7, 8]. The Yelp2018 dataset, which is the 2018 edition of the Yelp challenge, was published by X. Wang *et al.* [9] on GitHub[2]. The authors extracted item knowledge from the local business information network (e.g., category, location, and attribute) to construct the KG. The details on the dataset and the corresponding KG can be found in [9]. Following [7, 8], for each dataset, the ratio of training set, validation set, and test set is 6:2:2. Table 1 shows the statistics of the datasets and the KGs.

1 https://github.com/hwwang55/KGNN-LS/tree/master/data.

2 https://github.com/xiangwang1223/knowledge_graph_attention_network/tree/master/Data.

Table 1. Statistics of the three datasets

	MovieLens 20M	Last.FM	Yelp2018
# users	138,159	1,872	45,919
# items	16,954	3,846	45,538
# interactions	13,501,622	42,346	1,185,068
# entities	102,569	9,366	90,961
# relations	32	60	42
# KG triples	499,474	15,518	1,853,704

Evaluation Metrics. Three popular evaluation metrics [1], Precision at N (P@N), Recall@N (R@N), and F1-measure@N (F1@N), are used to evaluate the top-N recommendation performance (N = 5 or 10).

Model Implementation. The Python code of KGNN-LS was obtained from the GitHub webpage[3]. The code of iKGNN-LS was generated by modifying the Python code of KGNN-LS, specifically, by adding the implementation of the user-specific entity scoring function and the improved edge weights, as well as replacing the sum pooling aggregator with the max pooling one.

Hyperparameter Setting. Like [7, 8], in both iKGNN-LS and KGNN-LS, we set σ as *ReLU* for non-last-layers and *tanh* for the last-layer. We used grid search to select the hyperparameters for the two models. More specifically, just as in [7, 8], we selected the number K of sampled neighbors for entities in $\{2, 4, 8, 16, 32\}$, the dimension d of hidden layers in $\{4, 8, 16, 32\}$, the number L of layers in $\{1, 2\}$, the label smoothness regularizer weight λ in $\{0.01, 0.1, 0.5, 1.0, 1.5\}$, the L2-regularizer weight γ in $\{10^{-9}, 10^{-8}, 5\times10^{-8}, 10^{-7}, 5\times10^{-7}, 10^{-6}, 5\times10^{-6}, 10^{-5}, 5\times10^{-5}, 10^{-4}\}$, and the learning rate η in $\{10^{-4}, 2\times10^{-4}, 5\times10^{-4}, 10^{-3}, 2\times10^{-3}, 5\times10^{-3}, 10^{-2}, 2\times10^{-2}\}$. The resulting optimal hyperparameter settings for the three datasets are shown in Table 2.

Table 2. Hyperparameter settings for the three datasets

	MovieLens 20M	Last.FM	Yelp2018
K	16	8	32
d	32	16	16
L	1	1	1
λ	1.0	0.1	1.0
γ	10^{-7}	10^{-4}	10^{-9}
η	2×10^{-2}	5×10^{-4}	5×10^{-3}

[3] https://github.com/hwwang55/KGNN-LS.

3.2 Experimental Results

As in [8], each experiment was repeated 5 times, and the average performance is reported here. For the influence of edge weights on recommendation performance, Table 3 shows the top-N recommendation results on MovieLens 20M, where the figures in columns KGNN-LS and KGNN-LS-entity mean the results of the original KGNN-LS and the KGNN-LS that adds the user-specific entity scoring function, respectively, and the "Improvement (%)" figures refer to the percentages of performance improvement of KGNN-LS-entity over KGNN-LS. The results suggest that the KGNN-LS that adds the user-specific entity scoring function outperforms KGNN-LS in terms of all the metrics. This indicates that the edge weights determined jointly by personalized user preferences for relations and for entities can improve recommendation performance.

For the influence of pooling aggregators on recommendation performance, Table 4 shows the top-N recommendation results on MovieLens 20M, where the figures in columns KGNN-LS and KGNN-LS-max mean the results of the original KGNN-LS and the KGNN-LS that uses max pooling instead of sum pooling for neighborhood aggregation, respectively, and the "Improvement (%)" figures refer to the percentages of performance improvement of KGNN-LS-max over KGNN-LS. The results suggest that the KGNN-LS that uses max pooling outperforms KGNN-LS in terms of all the metrics. This indicates that the max pooling is better than the sum pooling in aggregating neighborhood in the recommendation context.

For the comparative experiment between KGNN-LS and iKGNN-LS, Table 5 shows the top-N recommendation results on the MovieLens 20M, Last.FM, and Yelp2018 datasets, where the "Improvement (%)" figures refer to the percentages of performance improvement of iKGNN-LS over KGNN-LS. The results suggest that on the three datasets, iKGNN-LS outperforms KGNN-LS in terms of all the metrics. This indicates the performance advantage of iKGNN-LS over KGNN-LS.

Table 3. Top-N recommendation results on MovieLens 20M (influence of edge weights)

Metrics	KGNN-LS	KGNN-LS-entity	Improvement (%)
P@5	0.1260	**0.1280**	1.59
P@10	0.0940	**0.0980**	4.26
R@5	0.0989	**0.0996**	0.71
R@10	**0.1550**	**0.1550**	0.00
F1@5	0.1108	**0.1120**	1.08
F1@10	0.1173	**0.1201**	2.39

Table 4. Top-N recommendation results on MovieLens 20M (influence of pooling aggregators)

Metrics	KGNN-LS	KGNN-LS-max	Improvement (%)
P@5	0.1260	**0.1280**	1.59
P@10	0.0940	**0.0990**	5.32
R@5	0.0989	**0.0994**	0.51
R@10	0.1550	**0.1607**	3.68
F1@5	0.1108	**0.1118**	0.90
F1@10	0.1173	**0.1225**	4.43

Table 5. Top-N recommendation results on MovieLens 20M, Last.FM, and Yelp2018

Dataset	Metrics	KGNN-LS	iKGNN-LS	Improvement (%)
MovieLens 20M	P@5	0.1260	**0.1290**	2.38
	P@10	0.0940	**0.1020**	8.51
	R@5	0.0989	**0.1022**	3.34
	R@10	0.1550	**0.1559**	0.58
	F1@5	0.1108	**0.1141**	2.98
	F1@10	0.1173	**0.1232**	5.03
Last.FM	P@5	0.0300	**0.0320**	6.67
	P@10	0.0280	**0.0300**	7.14
	R@5	0.0589	**0.0649**	8.53
	R@10	0.1223	**0.1329**	8.67
	F1@5	0.0399	**0.0429**	7.52
	F1@10	0.0456	**0.0491**	7.68
Yelp2018	P@5	0.0100	**0.0110**	10.00
	P@10	**0.0070**	**0.0070**	0.00
	R@5	0.0122	**0.0128**	4.92
	R@10	0.0184	**0.0195**	5.98
	F1@5	0.0110	**0.0118**	7.27
	F1@10	0.0101	**0.0103**	1.98

4 Conclusions

To overcome the weaknesses of KGNN-LS and learn user-specific item embeddings more effectively, in this paper we propose the improved iKGNN-LS model, which exploits user preferences for relations and for entities to jointly determine user-specific edge weights and uses max pooling instead of sum pooling to aggregate the most representative information of the neighborhood. Our comparative experiments of top-N recommendation on three datasets demonstrate the performance advantage of iKGNN-LS over KGNN-LS. Our future work will focus on further enhancing the iKGNN-LS model by integrating knowledge about users from social networks into the model.

References

1. Aggarwal, C.C.: Evaluating recommender systems. Recommender Systems, pp. 225–254. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29659-3_7
2. Cao, Y., Wang, X., He, X., Hu, Z., Chua, T.: Unifying knowledge graph learning and recommendation: towards a better understanding of user preferences. In: Proceedings of the World Wide Web Conference, WWW 2019, pp. 151–161. ACM (2019). https://doi.org/10.1145/3308558.3313705
3. Guo, Q., Zhuang, F., Qin, C., et al.: A survey on knowledge graph-based recommender systems. CoRR abs/2003.00911 (2020). https://arxiv.org/abs/2003.00911
4. Ma, W., Zhang, M., Cao, Y., et al.: Jointly learning explainable rules for recommendation with knowledge graph. In: Proceedings of the World Wide Web Conference, WWW 2019, pp. 1210–1221. ACM (2019). https://doi.org/10.1609/aaai.v33i01.33015329
5. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proc. IEEE **104**(1), 11–33 (2016). https://doi.org/10.1109/JPROC.2015.2483592
6. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, JMLR Workshop and Conference Proceedings, vol. 48, pp. 2014–2023. JMLR.org (2016). http://proceedings.mlr.press/v48/niepert16.html
7. Wang, H., Zhao, M., Xie, X., Li, W., Guo, M.: Knowledge graph convolutional networks for recommender systems. In: Proceedings of the World Wide Web Conference, WWW 2019 pp. 3307–3313. ACM (2019). https://doi.org/10.1145/3308558.3313417
8. Wang, H., Zhang, F., Zhang, M., et al.: Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, pp. 968–977. ACM (2019). https://doi.org/10.1145/3292500.3330836
9. Wang, X., He, X., Cao, Y., Liu, M., Chua, T.: KGAT: knowledge graph attention network for recommendation. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, pp. 950–958. ACM (2019). https://doi.org/10.1145/3292500.3330989
10. Xian, Y., Fu, Z., Muthukrishnan, S., Melo, G., Zhang, Y.: Reinforcement knowledge graph reasoning for explainable recommendation. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, pp. 285–294. ACM (2019). https://doi.org/10.1145/3331184.3331203
11. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: Proceedings of the 7th International Conference on Learning Representations, ICLR 2019. OpenReview.net (2019). https://openreview.net/forum?id=ryGs6iA5Km
12. Xu, W., Xu, Z., Ye, L.: Computing user similarity by combining item ratings and background knowledge from linked open data. In: Meng, X., Li, R., Wang, K., Niu, B., Wang, X., Zhao, G. (eds.) WISA 2018. LNCS, vol. 11242, pp. 467–478. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02934-0_43
13. Ye, Y., Wang, X., Yao, J., et al.: Bayes EMbedding (BEM): refining representation by integrating knowledge graphs and behavior-specific networks. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, pp. 679–688. ACM (2019). https://doi.org/10.1145/3357384.3358014