Learning Knowledge Embeddings with Prior Weights for Sparse Interaction Recommendation

Deqing Yang∗‡, Zikai Guo∗, Yanghua Xiao†

†Corresponding author, *School of Data Science, ‡School of Computer Science, Fudan University, Shanghai, China

Abstract—Knowledge-based recommendation models have exhibited their excellent performance in recent years. Most of these models encode knowledge into item embeddings through a graph embedding algorithm, which are useful for uncovering correlations between users and items. However, the graph embedding algorithms in these models neglect the different weights of various relations between items (entities), thus imprecise embeddings are learned resulting in unsatisfactory recommendation results. To address this problem, we propose a deep knowledge-based recommendation model which incorporates a novel graph embedding algorithm with prior relation weights, to learn precise item embeddings. Specifically, an HIN is first constructed based on the entities and relations from open knowledge graphs (KGs). Then, the embeddings of item vertices in the HIN are learned through seeking similar items in terms of various attributes (relations) with different prior weights. Next, the user representations are learned through user-tag-item relationships, based on which recommendation results are obtained by a multi-layer perceptron (MLP) fed with user presentations and item representations (embeddings). All the embeddings learned in our model are regarded as knowledge embeddings. The extensive experiments show that, our model outperforms the previous KG-based recommendation models with help of precise knowledge embeddings. Furthermore, it owns robust performance in the scenario of sparse user-item interactions, since it captures user preferences mainly based on the knowledge rather than observed user-item interactions.

Index Terms—knowledge embedding, heterogeneous information network, knowledge graph, recommendation

I. INTRODUCTION

As the most popular family of recommendation models, collaborative-filtering (CF) based models [1], [2] generally infer user preferences and item characteristics based on observed user-item interactions, e.g., purchase records, reviews or ratings, thus exhibit limited performance when observed user-item interactions are sparse. To alleviate this problem, more and more recommender systems resort to auxiliary (side) information to enrich the representations of users [3] or items [4]. Among them, knowledge graphs (KGs) provide one of typical side information to overcome the problem of sparse user-item interactions [5], [6].

The existing KG-based recommender systems can be mainly divided into two groups: path-based models and embedding-based models [7]. The first group of models usually construct a heterogeneous information network (HIN) consisting of users, items (entities) and relations distilled from open KGs at first. Then, the latent relationships between target users and candidate items are discovered through modeling the connected paths (meta paths) between them [5]. Comparatively, the second group of models try to learn the embeddings of users and items by a KG embedding algorithm, such as TransE [8] or Metapath2Vec [9], and then the recommendation results are computed based on user/item embeddings [10]–[12].

However, the graph embedding algorithms used in previous KG-based models do not utilize the knowledge sufficiently to generate precise item embeddings. Some algorithms [13], [14] do not distinguish different relation types (corresponding to different attribute types) between entities in KGs. Although the models in [9], [15] take into account relation types, they neglect that different relation types in fact indicate the relatedness between items to different extents. For example, more users may select their favorite movies based on movie actors rather than movie genres because the movies belonging to one genre are much more than the ones starring an actor. Fig. 1 displays a subgraph of movie HIN which is distilled from a KG. In the figure, we can find that the three movies both have relations to each other through their common attribute vertices, e.g., common director, actors or country. Many users who have watched movie Titanic may like to watch Inception rather than The Terminator if they are the fans of actor DiCaprio, although Inception and The Terminator both belong to science fiction movies. Therefore, common actor indicates the relatedness or similarities between two movies more obviously than common genre. Previous graph embedding algorithms neglect assigning different weights to different movie attributes when learning movie embeddings [9], [13], resulting in that the embeddings of Titanic and The Terminator have indiscriminated distances to the embedding of Inception. Thus the recommendation results generated based on such learned movie embeddings still have room for improvement.

To address above issue, we propose a novel KG-based recommendation model with knowledge embeddings which...
are learned based on the prior weights of different relations between item entities. Our model also belongs to embedding-based recommendation models with KGs. With the help of knowledge, our model still exhibits good performance when very sparse user-item interactions are observed. Without loss of generality, we introduce our model w.r.t. Douban\(^1\) movie recommendation of implicit feedback [16], [17]. Our model outperforms some existing KG-based recommendation models thanks to the following strategies:

1) In our proposed HIN embedding algorithm, we not only consider different types of relations between items (entities), but also propose a method to compute the prior weights for different relation (attribute) types which quantify various relations’ importance on seeking similar items. Therefore, our model can generate item embeddings of finer granularity than previous graph embedding algorithms, resulting in better recommendation performance.

2) In the scenario of sparse user-movie interactions, a user in our model is specified by tag representations instead of observed user-movie interactions, which alleviates the problem of sparse interactions. Furthermore, our model can accomplish recommendation even for the cold-start users/items without any user-item interactions.

The contributions of our work are summarized as follows:

1) We propose a weighted HIN embedding algorithm to learn more precise item embeddings. It seeks similar items in terms of different item attributes assigned with well-computed prior weights, instead of issuing random walks as traditional graph embedding models.

2) To ensure our model’s robust recommendation performance in the scenario of sparse user-item interactions, we design a solution of representing a user by summing up his/her tag representations with attentions, rather than direct pooling the representations of the user’s historical interacted items. Our model can further improve its performance when adequate user-item interactions are observed since its user representations also encode user preferences inferred from observed interactions.

3) The extensive experiments demonstrate the superiority of our HIN embedding algorithm with prior weights on learning precise item embeddings. Furthermore, our model’s robust recommendation performance in the scenario of sparse user-item interactions is also justified by our evaluations.

The rest of this paper is organized as follows. We present the design details of our model in Section 2, and show our experiment results in Section 3. In Section 4, we introduce related work and conclude our work in Section 5.

II. METHODOLOGY

In this section, we describe the details of the proposed recommendation model and the overview of its framework is displayed in Fig. 2. Our model can be divided into two steps. The first step is to learn movie embeddings through embedding an HIN which is constructed based on substantial movie knowledge from KGs. This step is the prerequisite of the next step. In the second step, our framework represents a user by the attentive sum of his/her tag representations, which are also derived from movie embeddings learned in the first step and encode the user’s preference precisely. The final recommendation results are generated by an MLP fed with user representations and movie representations.

---

\(^1\)https://movie.douban.com, it is a famous Chinese review website for movies, books, music and etc.

\(^2\)http://kw.fudan.edu.cn/cndbpedia, it is a large scale Chinese KG containing more than 16.8 million entities and 0.2 billion relations.
function that maximizes the probability of observing a network neighborhood \(v\) for a given vertex \(u\) conditioned on its feature embeddings \(\theta\):

\[
\arg \max_{\theta} \prod_{v \in U} \prod_{v \in C(u) \land v \neq u} p(v|u; \theta)
\]

where \(p(v|u; \theta)\) is the probability of observing \(v\) given \(u\), and \(C(u)\) is \(u\)'s neighborhood, i.e., the context of \(u\). Obviously, the prerequisite for solving the objective in Eq. 1 is seeking \(C(u)\). Traditional graph embedding algorithms [13], [14] use random walks rooted at \(u\) under a length constraint \(l\), namely \(w_{k_u}\), to generate \(C(u)\). Accordingly, the \(C(u)\) derived in this way consists of all vertices traversed by \(w_{k_u}\). From the perspective of network topology, the vertices in \(C(u)\) can be regarded as the ones similar to \(u\) (or have relationships to \(u\)). In our scenario, we can start random walks from movie vertices and retain movie vertices in \(C(u)\), since we require movie embeddings which are the basis of our recommendation model. The probability with which a random walk is started from \(u\) is proportional to \(u\)'s popularity. For Douban movie recommendation, \(u\)'s popularity can be quantified as its rating count.

In order to decrease model’s computational complexity, negative sampling [19] is adopted in which \(p(v|u; \theta)\) is replaced by a Sigmoid function as

\[
p(v|u; \theta) = \sigma(e_u \cdot e_v) = \frac{1}{1 + e^{-(e_u \cdot e_v)}}
\]

where \(e_u\) is \(u\)'s embedding vector. If we use \(y_{iu}^v \in \{0, 1\}\) to indicate whether \(v\) is in \(C(u)\) or not, we should maximize the following objective function given \(u\) according to Eq. 1

\[
\mathcal{O}_u = \prod_{v \in C(u) \land v \neq u} \left\{ \sigma(e_v \cdot e_u)^{y_{iu}^v} \prod_{i=1}^{M} \mathbb{E}_{y_{iu}^v \sim p} [1 - \sigma(e_v \cdot e_u)]^{1-y_{iu}^v} \right\}
\]

where \(\mathcal{P}\) is the vertex distribution computed based on movie rating count, from which a negative vertex \(v_i\) \((v_i \notin C(u))\) is drew for \(M\) times. This objective indicates that \(\sigma(e_v \cdot e_u)\) should be as large as possible if \(y_{iu}^v=1\), or be as small as possible if \(y_{iu}^v=0\), which is proportional to the similarity between \(v\) and \(u\).

B. Seeking Similar Movies through Various Relations with Prior Weights

As introduced before, the key step of learning movie embeddings based on the embedding model is collecting similar movies, i.e., generating \(C(u)\) for a given movie \(u\) in Eq. 1. In general, two movies’ similarity is exhibited through their common attribute vertices, such as The Terminator and Titanic share common director/writer James Cameron, Titanic and Inception share common actor DiCaprio. In fact, these common attribute vertices correspond to different relations between two movies, and different relations should have different weights (extents) on indicating movie similarity. In this subsection, we introduce a method of seeking the movies similar to a given movie in terms of various relation assigned with different prior weights. Our experiments justify that our proposed method is more effective on collecting similar movies than those random walk based models [9], [13].

At first, we use the probability \(p(u, v)\) to quantify the extent to which a movie \(v\) is similar to another movie \(u\). Specifically, we compute \(p(u, v)\) as

\[
p(u, v) = \sum_{a} w_a \text{sim}_a(u, v)
\]

where \(a\) represents a certain movie relation (attribute) and \(w_a\) is \(a\)'s normalized prior weight to quantify \(p(u, v)\). We compute \(w_a\) based on the observations in the HIN. \(\text{sim}_a(u, v)\) is the similarity between \(u\) and \(v\) in terms of \(a\). Next, we will introduce how to compute \(w_a\) and \(\text{sim}_a(u, v)\) in turn.

1) Obtaining Prior Weights of Relations: In total, we take into account five relations between two movies, i.e., the relations through common actor, director, writer, genre and semantically related synopsis. In general, one movie has a group of actors, one or two directors/writers, and one synopsis. In addition, one movie may belong to multiple genres. The quantification of \(w_a\) is based on an intuition that, a relation contributes less to seeking similar movies if an attribute vertex corresponding to this relation connects more movies. For example, the movies starring a certain actor are much less than the movies belonging to a certain genre. More users recognize their favorite movies in terms of actors instead of genres. Consequently, common actor vertices indicate the similarity between two movies more evidently than common genre vertices. Based on this intuition, we compute \(w_a\) as follows. Given a movie attribute \(a\) which corresponds to a certain movie relation, we first denote an entity member of \(a\) as \(i_a\), and the movie involving \(i_a\) as \(m_a\). The average number of movies involving \(i_a\) is \(\overline{m_a}\) which is a certain movie relation (attribute) and \(\text{sim}_a(u, v)\) is the similarity between \(u\) and \(v\) in terms of \(a\). Next, we will introduce how to compute \(w_a\) and \(\text{sim}_a(u, v)\) in turn.

2) Computing Movie Similarities in Terms of Relations: Before introducing the computation of \(\text{sim}_a(u, v)\), we first define the affinity score of an actor/director/writer to a movie. Specifically, given a member \(i\) of actor/director/writer, we suppose \(u\) is a movie involving \(i\), i.e., \(u \in \{m_i\}\), and \(u\)'s genre set is \(G(u)\). Then, \(i\)'s affinity score to \(u\) is computed as

\[
s(i, u) = \log \frac{|M|}{|\{m_i\}|} \times \frac{\{m_i | G(m_i) \cap G(u) \neq \emptyset\}}{|\{m_i\}|}
\]

where \(M\) is the whole movie set. The rationale of \(s(i, u)\)'s computation is two-fold: 1) The first factor take the same effects as IDF. 2) The second factor implies that, \(i\) is more informative for \(u\) if \(i\) is involved in more movies which have the same genres as \(u\)'s genres. Next, we propose \(\text{sim}_a(u, v)\)'s computation in terms of different movie relations.

a) Actor: Given two movies \(u\) and \(v\), we use \(U\) and \(V\) to represent the actor set of \(u\) and \(v\), respectively. Then,

\[
\text{sim}_{\text{actor}}(u, v) = \frac{\sum_{i \in U \cap V} s(i, u) s(i, v)}{\sum_{i \in U} s(i, u) + \sum_{j \in V} s(j, v)}
\]
where \( k_i \) is actor \( i \)'s rank in movie \( u \)'s actor list. Note that we set \( k_i = 2 \) if \( i \) is the first actor of \( u \). The rationale of such setting is that, most movies have two leading actors and the top ranked actors are more informative for a movie than the ones behind.

b) Director/Writer: If \( a \) is the attribute of director or writer, we have

\[
sim_{\text{director/writer}}(u, v) = \frac{\sum_{i \in U \cap V} s(i, u) + \sum_{j \in U \cap V} s(j, v)}{\sum_{i \in U} s(i, u) + \sum_{j \in V} s(j, v)}
\]

where \( U \) and \( V \) are the director/writer set of \( u \) and \( v \), respectively. We neglect a director/writer's rank in a movie’s director/writer list since most movies only have one or two director/writers.

c) Genre: For movie genre, we have

\[
sim_{\text{genre}}(u, v) = \frac{2 \times |G(u) \cap G(v)|}{|G(u)| + |G(v)|}
\]

d) Synopsis: Beside above attributes, the semantic similarities between movie synopses are also very important to infer movie relatedness. Therefore, given two movie \( u \) and \( v \), we first fetch their synopses from CN-DBpedia, and then use a Doc2Vec [20] model pre-trained by Wikipedia corpus to generate the synopsis vectors of \( u \) and \( v \). At last, we use the two vectors' cosine distance as \( \text{sim}_{\text{synopsis}}(u, v) \). In addition, we set \( w_{\text{synopsis}} \) equal to \( w_{\text{actor}} \) which is the largest for above four attributes.

Besides aforementioned five attributes, other movie attributes such as poster and trailer can also be used to discover movie similarity based on some complex metrics. After we obtain \( p(u, v) \) in terms of all attributes, we construct \( C(u) \) according to the following steps:

1) Given a movie \( u \), we first set \( |C(u)| \) to the logarithm of \( u \)'s rating count. The rationale of setting \( |C(u)| \) proportional to \( u \)'s popularity is as follows. According to the principle of the embedding model, a movie \( u \)'s embedding will be learned better if \( |C(u)| \) is larger. From the perspective of overall recommendation performance, those popularity movies have more chances to be recommended to users, thus they deserve being represented with more precise features than the unpopular movies.

2) Then, we select \( |C(u)| \) similar movies with top \( p(u, v) \) to constitute \( u \)'s similar movie group.

3) We repeat for \( |C(u)| \) times to draw a \( v \) from \( u \)'s similar movie group to join \( C(u) \) according to \( p(u, v) \).

C. Generating Attentive User Representations

In most CF-based recommendation models with knowledge [6], [21], a user is generally represented by the combination of the embeddings of his/her favorite items discovered from observed user-item interactions, such as watched movies and bought products. Although these favorite items indicate a user’s personal preference, such user representations suffer from the scenario of sparse observed interactions. It motivates us to resort to other data other than user-item interactions for representing users.

In Douban, each movie and user are both labeled with a set of tags, which not only characterize a movie but also specify a user’s preference. It inspires us to generate user representations by using user tags. To this end, tag embeddings should be obtained at first. As illustrated in Fig. 2, a tag’s embedding is obtained by an average pooling for the embeddings of the movies labeled by this tag. Then, a simple operation of representing a user based on his/her tag embeddings is also average pooling. However, we argue that different tags of a user specify the user to different extents. For example, if a user has watched three movies labeled by tag 'war' and one movie labeled by tag 'comedy', we believe that the user possibly likes war movies more than comedy movies. It reminds us to assign ‘war’ with a weight bigger than ‘comedy’ to represent this user. Therefore, a given user \( u \)'s attentive representation is defined as

\[
r^u = \sum_t w_{u,t} r^i_t
\]

where \( r^i_t \) is the embedding of \( u \)'s tag \( t \), and \( w_{u,t} \) is \( t \)'s attention weight on representing \( u \). We use the following softmax function to compute \( w_{u,t} \),

\[
w_{u,t} = \frac{\exp(\beta r^u_t \cdot r^i_t)}{\sum_{t'} \exp(\beta r^u_{t'} \cdot r^i_t)}
\]

where \( \beta \) is a scaling factor and empirically set to 10 [22]. We define \( r^u_t \) as \( u \)'s personal preference which is computed by

\[
r^u_t = \sum_i w_{u,t} r^i_t
\]

where \( r^i_t \) is the embedding of \( u \)'s watched movie \( i \) found from observed user-item interactions. \( w_{u,t} \) is the normalized weight of \( i \) to \( u \), which is proportional to \( u \)'s rate score on \( i \). If we do not observe any \( u \)'s watched movies from user-item interactions, we set \( r^u_t \) to the average of all popular movies’ embeddings. In our experiments, the popular movies are filtered as the movies having more than 10,000 rating count. The rationale of Eq. 11 is that, the tags of which the representations are more closer to \( u \)'s personal preference contribute more to \( u \)'s representation.

Such operation of generating user representations has another merit. When more user-item interactions are observed, \( r^u_t \) captures \( u \)'s preference more precisely, resulting in more accurate \( w_{u,t} \) and \( r^i_t \). Thus, the final recommendation performance can be further improved further. It will be justified by the experiment results shown in the next section.

D. Accomplishing Recommendation

At last, in order to predict a given user \( u \)'s implicit feedback to a candidate movie \( m \), i.e., judge whether \( u \) will watch \( m \) or not, \( u \)'s representation and \( m \)'s representation are concatenated and then fed into a multi-layer perceptron (MLP for short) network. In Fig. 2, \( \tilde{y}_{um} \) is the output score (probability) of MLP based on which we can not only predict \( u \)'s implicit feedback to \( m \) but also get top-n recommendation results if all candidate movies are ranked according to the score. In the scenario of sparse user-movie interactions, although \( r^u_t \) is not accurate to specify \( u \), our model exhibits good performance mainly thanks to \( r^i_t \). Even when no user-movie interactions are observed, it can still accomplish recommendation if we
directly set $r^u$ to the average of $u$’s tag embeddings, and then compute the distance between $r^u$ and $m$’s representation instead of training the MLP based on observed interactions.

Since a KG generally contains substantial entities and relations related to various items, including movies, books, musics and etc., our model can be extended to other recommendation domains. Even for the case that tags do not exist in the domain, our model still works if we set $r^u = r^u_p$, so long as some user-item interactions can be observed to train the MLP.

III. EXPERIMENTS

A. Experiment Settings

1) Dataset Description: In order to collect user-movie interaction data in Douban, we randomly selected 5,000 Douban users who have rated 15 movies at least. Then, we fetched these users’ profiles (including tags), as well as their rating scores on their watched movies. From the movies rated by the 5000 users, we filtered 42,000 movies. We also got all these movies’ tags labeled by Douban users. In total we fetched 89,909 unique tags for these users and movies.

We also distilled the knowledge about Douban movies from CN-DBpedia [18]. The relevant data can be fetched through open APIs in the website of Knowledge Works1. The obtained attribute information about Douban movies includes directors, actors, writers, genres and synopsis. All datasets used in our experiments and the codes of our model can be obtained from https://doi.org/10.7910/DVN/WCXPQA.

2) Sample Collection: The task addressed in this paper belongs to implicit feedback [5], [16], [17]. In formal, the feedback value is 1 if we observe a user $u$ has rated (watched) a movie $m$ from user-movie interactions. However, it does not mean $u$ actually likes $m$. Meanwhile, a value of 0 does not necessarily mean $u$ does not like $m$, it is possibly because $u$ is not aware of the movie. Nevertheless, an observed interaction $u \rightarrow m$ at least reflects $u$’s interest on $m$, the unobserved interactions can be just missing data. In our training/test sets, we implicitly recognize the movies with $u$’s feedback of value 1 as $u$’s positive movies, and recognize the rest movies as $u$’s negative movies.

As we claimed before, our model is designed also towards the recommendation with sparse user-item interactions. To justify our model’s performance in such scenario, we adopted the method used in [12] to simulate some scenarios of sparse interactions. Specifically, we first selected some user-movie interactions in random from the whole set according to a certain proportion, namely $p$. These selected interactions were used to construct training set for training the MLP in our framework and generating user’s personal preference representation. Obviously, small $p$ results in sparse known user-movie interactions. Accordingly, traditional CF-based algorithms can only utilize the interactions in training set to specify users and movies. To train the MLP better, for each user in training set, we also randomly added his/her negative movies into training set, of which the number is 20 times of the number of his/her positive movies. In the rest user-movie interactions, we selected those users having at least 10 positive movies as valid test users, and further coupled 200 negative movies not existing in training set with them, to constitute our test set.

3) Competitors:

a) LOD: This baseline [6] first represents each movie with several attribute vectors in terms of attribute similarities which are distilled from KGs. Then, the similarity between two movies is computed as the weighted average of their attribute vectors’ similarities (cosine distances). At last, a user’s preference on a candidate movie is inferred according to item-based CF scheme.

b) HIN: As LOD, this baseline [5] also infers a user’s preference on a candidate movie based on item-based CF. As our model, it also constructs a HIN including movies and various attributes, and then movie similarity is measured by counting different meta-paths linking two movies. Although HIN and LOD both seek movie similarities through KG (HIN), they accomplish recommendation depending on observed user-movie interactions.

c) NCF: Neural Collaborative Filtering [16] has been proved to be a powerful DNN-based CF framework consisting of a GMF (generalized matrix factorization) layer and an MLP. Both GMF and MLP are fed with user and item representations initialized in random. NCF parameters are learned based on obtained user-item interactions. Thus it is doomed to be sensitive to sparse interactions as other CF-based models.

d) DW: In order to learn movie representations, this graph embedding model issues random walks in the HIN as DeepWalk [13] to generate $C(u)$ in Eq. 3 for a given movie $u$. Note that any two movies can not be connected through a synopsis vertex since they have no common synopsis. Comparatively, our model seeks $C(u)$ through synopsis similarity, thus more latent relationships between movies are found.

e) M2V: This baseline is Metapath2Vec [9], which also issues random walks to collect related vertices (movies) in the HIN for learning movie representations, but the walks are biased according to different meta-paths. Since M2V has been proved more effective than other state-of-the-art graph embedding models such as [14], [23], [24], we believe our model also outperforms these models if our model is justified better than M2V.

f) KE/T/P: We use KE (knowledge embedding) to denote learning movie embeddings by our HIN embedding algorithm with prior weights. According to the computation of Eq. 5, we got $w_{au}$ of ‘actor, synopsis, director, writer, genre’ as ‘0.36, 0.36, 0.1, 0.16, 0.02’. We also use T and P to denote the operation of generating user representations by $r^u$ and $r^u_p$, respectively. Accordingly, our proposed model can be denoted as KE+T, and is adjusted to KE+P if no user tags are obtained. We compared KE+T with KE+P in our experiments to emphasize the significance of utilizing user tags in the scenario of sparse user-item interactions. Furthermore, we also propose a variant of our HIN embedding algorithm, namely KE0. In K0, each movie relation’s weight is set to 1/5 instead of the value computed by Eq. 5. We compared KE with KE0 to emphasize the significance of well-computed prior weights to seeking similar movies in HIN embedding.

B. Evaluation Results

1) Tuning Embedding Dimension: We sought the optimal dimension of user/movie embeddings (representations)
through tuning experiments. Since we propose our model addressing the recommendation with sparse user-item interactions, we mainly concern our model’s recommendation performance in the scenario of $p=0.1\%$, where only 16% of test users could be found from observed interactions, i.e., training set. Table I lists top-10 movie recommendation performance scores of KE+T with different embedding dimensions. The results show that 200 dimension is the best setting, which was also confirmed by the results in the scenarios of other $p$s. We do not list other results due to space limitation. Thus, we set the dimension of user/movie representations to 200 when we ran relevant models in the following comparison experiments.

<table>
<thead>
<tr>
<th>dim.</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prc. @10</td>
<td>0.2540</td>
<td>0.2684</td>
<td>0.2698</td>
<td>0.2871</td>
<td>0.2666</td>
</tr>
<tr>
<td>AP@10</td>
<td>0.1421</td>
<td>0.1612</td>
<td>0.1579</td>
<td>0.1701</td>
<td>0.1595</td>
</tr>
<tr>
<td>nDCG@10</td>
<td>0.2821</td>
<td>0.3067</td>
<td>0.3030</td>
<td>0.3201</td>
<td>0.3044</td>
</tr>
</tbody>
</table>

**TABLE I**

Top-10 movie recommendation performance of KE+T with different dimensions of user/movie embeddings when $p=0.1\%$.

**Fig. 3.** Attribute similarities between *Skyfall* and its top 5 similar movies.

2) Effectiveness on Seeking Similar Movies: We compared the performance of our proposed KE with KE0, LOD, HIN, DW and M2V on seeking movie similarity because it is the prerequisite of the downstream recommendation. Table II lists 3 target movies and their top 5 closest movies in terms of movie similarity computed by the compared methods. From the results we find that, KE uncovers more movies similar to the target movies. For example, KE discovers more *007 (James Bond)* movies w.r.t. *Skyfall* than its rivals. The results emphasize that our proposed HIN embedding algorithm can learn more precise movie embeddings than the competitors. KE’s advantage over KE0 also highlights the significance of different prior weights for various movie relations.

We also present a case study of Fig. 3 to investigate the capability of knowledge embeddings on indicating different relations between items. The figure displays the attribute similarities (Eq. 7 ～ 9) between movie *Skyfall* and its top 5 similar movies discovered by the cosine distances between movie embeddings. It shows that the top 3 movies are found mainly through their common actors and semantically related synopses. Comparatively, *Road to Perdition* is found mainly through its common director and related synopsis, although it is not a 007 movie. Based on these uncovered similar items with various relations, the diversity and interpretability of recommendation results are improved, which are generally the key metrics measuring recommendation performance.

3) Effectiveness on Movie Recommendation: In order to demonstrate our model’s outperformance no matter how many user-item interactions are observed, we compared all models’ top-10 recommendation performance in the scenarios of $p=0.1\%$～$40\%$. We did not concern the scenarios of $p > 40\%$ because each test user can be observed from the interactions in training set when $p=40\%$, implying that user-item interactions are adequate to infer all users’ preferences sufficiently. Fig. 4 shows all models’ performance scores averaged on all test users. From the figure we find that KE+T outperforms its rivals in almost all scenarios. KE/KE0+T’s superiority over other models is very evident during $p=0.1\%$～$20\%$. As $p$ increases, NCF+T and KE+P catch up with KE+T gradually. Comparatively, KE+T keeps its superiority over DW+T and M2V+T consistently in all scenarios. Although LOD and HIN also improve their performance as $p$ increases, they fall far behind other models. Although KE0+T’s performance is very close to KE+T, its weakness is perceptible especially during $p=10\%$～$40\%$.

4) Discussions: Based on evaluation results, we discuss the reasons causing the differences between compared models.

a) KE’s advantage over DW and M2V shows that, taking into account the prior weights of different relations between items is more effective on seeking similar movies than generic random walks (DW) and the random walks along with different types of meta-paths (M2V). Another important reason is KE takes into account the semantic relatedness of movie synopsis. Moreover, the better learned movie embeddings also promote the downstream recommendation because DW+T and M2V+T are defeated by KE+T, although they both use the same operations of representing users and accomplishing recommendation.

b) KE+T and KE+P outperform NCF+T especially in the scenario of sparse interactions, showing that learning precise movie embedding is very significant for sparse interaction recommendation. NCF+T performs most weakly when $p=0.1\%$～$1\%$, implying that NCF’s performance is very sensitive to observed user-item interactions.

c) Although LOD and HIN also improve their performance as $p$ increases, the improvement is not so remarkable as their rivals. It is not only due to their poor performance on seeking similar movies, but also because their rivals both use an MLP which is more powerful on computing the matching degree between the target users and candidate items.

d) KE+T’s superiority over KE+P’s is significant in the scenarios of $p=0.1\%$～$20\%$, showing that user tags are very effective on specifying user preferences when observed user-movie interactions are sparse. However, KE+P outperforms KE+T a bit when $p=40\%$, implying that only considering user’s personal preference is enough for obtaining good recommendation performance when sufficient user-item interactions are observed. It also inspires us to directly add $r_p^u$ into $u$’s representation for the recommendation of sufficient interactions. For example, we can use the concatenation of $r_p^u$ and the vector generated by Eq. 10 as $u$’s final representation.

### IV. RELATED WORK

1) Representation Learning: As one family of feature engineering, representation learning has been extensively studied in the community of machine learning. Most related works
focus on learning the representations of nodes in networks or graphs [14], [25] rather than the objects in recommendation domain. Graph representation learning can be traced back to the usage of latent factor models for network analysis and graph mining tasks [26]. However, this family of research concentrates on factorizing matrix/tensor format of a network, of which the computational cost of decomposing a large-scale matrix/tensor is usually very expensive, causing it neither practical nor effective for addressing tasks in big networks. Swami et al. [9] proposed a scalable representation learning model for HIN based on the Skip-gram model [19], [27], but they neglect the variant weight of different meta-path towards different application scenarios, e.g., measuring the similarity between two movies, which is the focus of this paper. Fu et al. proposed HIN2Vec [15] model to embedding a HIN, which also uses random walks to collect model samples and generates representations for vertices and edges.

2) Knowledge-based Recommendation: In early research stage of Knowledge-based recommendation [28], the knowledge includes item’s specific attributes which are used as constraints of filtering out a user’s preferred items. As more and more KGs emerge, many researchers found that the rich knowledge in KGs can be used as auxiliary information to overcome cold-start problem. Similar to our work, Noia et al. [6] also utilized knowledge from open KGs to measure movie similarity for improving recommendation performance. More than using structured knowledge (movie attributes) in KGs, Zhang et al. [17] further fed text and image knowledge of movies in deep learning framework to achieve powerful recommendation performance. [5] also uses KGs to construct a HIN for movies, but it measures the relatedness between movies only through the volume of meta-paths, which is only explicit feature less effective than the feature learned by embedding model. [21] applies Node2Vec [14] to learn user/item representations according to different relations between entities in KGs. But it is still a CF-based method of poor performance when user-item interactions are sparse. More recently, Yang et al. [12] utilized Metapath2Vec [9]
to generating movie embeddings based from KGs, based on which a GAN-based model [29] was constructed to obtain better performance. Metapath2Vec has been proven not good as our HIN embedding model in this paper.

3) Embedding-based Recommendation: Since word embeddings can well represent content-based features in semantics, the embedding model has been imported into some content-based recommender systems. Many of previous works tried to embed the keywords extracted from documents instead of tags, to enrich the content-based specification of users/items. The model in [30] maps items into textual contents through embedding the keywords from Wikipedia documents, for the recommendations on MovieLens and DBbook datasets. Shin et al. [31] employed Word2Vec to capture the embeddings of tags which facilitate inductive matrix completion (IMC) in recommending Tumblr blogs, but their tags are the keywords distilled from blog texts. In addition, some non-textual features can also be embedded to make effective recommendation, such as the user check-ins of Foursquare [32] and user clicked URLs [33]. In general, the keywords from free texts are more noisier and thus not so effective as well-defined tags on specifying users/items. Yang et al. [34] used the embedding vectors of user tags which are also learned based on Skip-gram model, to accomplish the recommendation of sparse user-item interactions. But they did not utilize KGs to enrich item representations.

V. Conclusion

Towards the recommendation with sparse user-item interactions, we propose a knowledge-based recommendation model in this paper. In order to learn better item embeddings which are the prerequisite of downstream recommendation, we design a novel HIN embedding algorithm with prior relation weights to seek similar items. Furthermore, to tackle the challenge of sparse user-item interaction, our framework combines a user’s interacted item embeddings and tag embeddings to represent the user, rather than direct synthesizing the embeddings of historical interacted items as other CF-based algorithms. Extensive experiments justify our HIN embedding algorithm’s superiority over previous graph embedding algorithm on learning item embeddings, thus better recommendation performance is also achieved.

REFERENCES