Real-time Social Recommendation Based on Graph Embedding and Temporal Context

Peng Liu[a,*], Lemei Zhang[a], Jon Atle Gulla[a]

[a]*Department of Computer Science, NTNU, 7491, Trondheim, Norway*

Abstract

With the rapid proliferation of online social networks, personalized social recommendation has become an important means to help people discover their potential friends or interested items in real-time. However, the cold-start issue and the special properties of social networks, such as rich temporal dynamics, heterogeneous and complex structures, render the most commonly used recommendation approaches (e.g. Collaborative Filtering) inefficient. In this paper, we propose a novel dynamic graph-based embedding (DGE) model for social recommendation which is capable of recommending relevant users and interested items. In order to support real-time recommendation, we construct a heterogeneous user-item (HUI) network and incrementally maintain it as the social network evolves. DGE jointly captures the temporal semantic effects, social relationships and user behavior sequential patterns in a unified way by embedding the HUI network into a shared low dimensional space. Then, with simple search methods or similarity calculations, we can use the encoded representation of temporal contexts to generate recommendations. We conduct extensive experiments to evaluate the performance of our model on two real large-scale datasets, and the experimental results show its advantages over other state-of-the-art methods.

Keywords: Real-time, Social recommendation, Heterogeneous social network, Graph embedding, Temporal context

1. Introduction

With the rapid development of Web 2.0 and smart mobile devices, online social networks have proliferated and are still promptly growing. According to Twitter statistics, the number of users is estimated to have surpassed 300 million generating more than 200 million tweets per day[1]. Faced with the abundance

*Corresponding author

Email addresses: peng.liu@idi.ntnu.no (Peng Liu), lemei.zhang@idi.ntnu.no (Lemei Zhang), jon.atle.gulla@idi.ntnu.no (Jon Atle Gulla)

[1]https://blog.twitter.com/2011/200-million-tweets-per-day, accessed: March 15, 2017.

of user generated content, a key issue of social networking services is how to help users find their potential friends or interested items that match the users' preference as much as possible, by making use of both semantic information and social relationships. This is the problem of personalized social recommendation.

Generally, different techniques used in building personalized recommender systems are mainly divided into three categories: collaborative filtering, content-based filtering and hybrid system (Aggarwal, 2016). Although previous techniques have been shown to be effective to some extent, there still exist two major challenges in front of online social networks. First, the complex structures in the social network need to be properly mined and exploited by algorithms. Second, these networks contain millions or even billions of edges making the problem very difficult computationally. For example, the efficiency of classic item-based k nearest neighbor (KNN) recommendation algorithms is largely limited by the construction of the KNN graph (Deshpande and Karypis, 2004). Matrix factorization involves eigen-decomposition of the data matrix which is expensive and usually with approximation calculation (Rendle and Schmidt-Thieme, 2010). Therefore, it is crucial to handle large-scale heterogeneous networks for social recommender system.

In recent years, there have been numerous studies exploiting different types of relationships in heterogeneous networks (Kouki et al., 2015; Shi et al., 2015; Sun and Han, 2013; Yu et al., 2014) to improve the quality of recommendations. However, considering the dynamic nature of social network, almost all existing social recommendation methods are incapable of supporting real-time recommendation principally, and they would suffer from the following three drawbacks: 1) Delay on model updates caused by the expensive time cost of re-running the recommender model. 2) Disability to track changing user preferences due to the fact that latest entries used for updating recommendation models are often overwhelmed by the large data of the past. 3) Cold start problem becomes even more severe in online social networks as the new users and new items will join in the recommender system constantly over time. Some online learning algorithms address this problem by keeping a representative sample of the data set in a reservoir to retrain the model (Diaz-Aviles et al., 2012), which however is not appropriate for large streaming data set. To avoid this problem, some other online algorithms propose to update the model based solely on the current observation (Vinagre et al., 2014), at the cost of reducing the quality of recommendations.

In this work, our goal for social recommendation is to provide real-time and accurate recommendation services for users in large-scale heterogeneous networks. Specifically, it demonstrates three requirements. First of all, the recommender system needs to produce accurate recommendations for users. Second, the model should be updated in real-time to capture users' instant interests and social network evolution in very short delay. Third, the processing needs to be executed in parallel, i.e., scalable to handle large amounts of computations.

To fulfill the aforementioned goals, we propose a novel dynamic graph-based embedding (DGE) model which can effectively recommend relevant users and interested items in real-time. Inspired by recent progress in network represen-

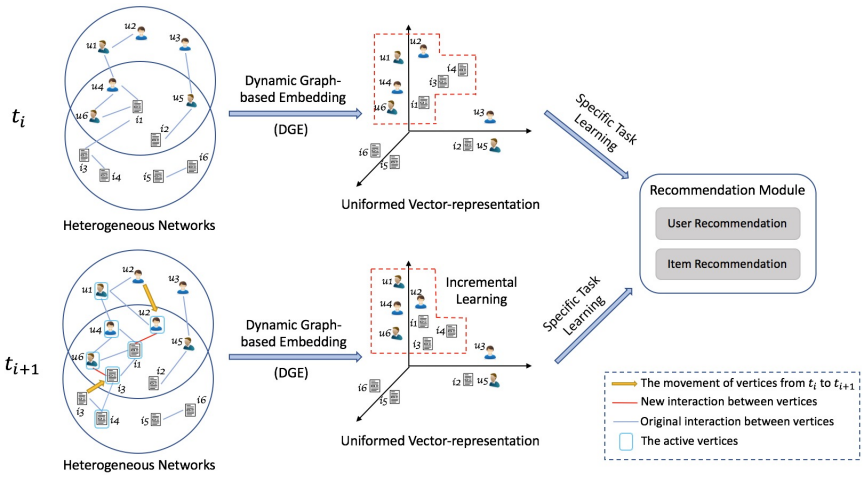


Figure 1: The flowchart of dynamic graph-based embedding framework.

tation learning and deep learning (Mikolov et al., 2013b; Perozzi et al., 2014; Tang et al., 2015b), we propose to use the distributed representation method for modeling online social networks. Specifically, we construct a heterogeneous user-item (HUI) network, in which the two types of vertices represent users and various items and the three types of edges respectively characterize the semantic effects, social relationships and user behavior sequential patterns. Based on the differential behaviour represented among continuous time slots, the HUI network is incrementally maintained as the social network evolves. Then, an incremental learning algorithm is applied to embed the HUI network into low-dimensional vector spaces, in which the proximity information of each vertex is encoded into its learned vector representation. Afterwards, we use the learned representations of vertices with some simple search methods or similarity calculations to conduct the task of social recommendation.

Fig. 1 illustrates the idea of dynamic graph-based embedding framework. To summarize, this paper makes the following contributions:

- We propose a dynamic graph-based embedding model that integrates the temporal semantic effects, social relationships and user behavior sequential patterns into the process of network embedding. To the best of our knowledge, this work is the first to address real-time social recommendation by a network representation learning approach.

- We devise a transition probability matrix P for the complex HUI network to capture the semantic effect of different edge types. Based on this, an asynchronous parallel stochastic gradient descent method is proposed to allow horizontally scaling the algorithm for large-scale social networks and improve the efficiency of the inference.

- To speed up the process of producing top-k recommendations from large-

scale social media streams, we develop an efficient query processing technique by extending the Threshold Algorithm (TA) (Fagin et al., 2003).

- We conduct extensive experiments to evaluate the performance of our model on two real large-scale datasets. The results show the advantages of our method for social recommendation in comparison with state-of-the-art techniques.

The remainder of the paper is organized as follows. Section 2 introduces the related work. In section 3, we formally define our problem and give the definition of each source for the heterogeneous network. Section 4 presents our new model. We describe the data sets, comparative approaches and the evaluation criteria we use in section 5. Section 6 shows our experiment results. Finally, we present the conclusions and future work in Section 7.

2. Related Work

2.1. Social Recommender System

Most traditional social recommendation models such as matrix factorization-based models (Koren, 2008), graph-based models (Aggarwal et al., 1999) and latent semantic models (Hofmann, 2004) etc, deal with homogeneous objects or separately deal with different types of objects. In recent years, the flourish of the heterogeneous social networks provides a new environment for recommendation targets. Kouki et al. (2015) proposed a hybrid approach, HyPER (Hybrid Probabilistic Extensible Recommender), to incorporate and reason over a wide range of information sources. Sun and Han (2013) explored the meta structure of the heterogeneous information network to boost similarity searching and other mining tasks. Shi et al. (2015) explored a weighted heterogeneous information network and weighted meta path based recommender system (SemRec) to predict the rating scores of users on items. To the best of our knowledge, existing heterogeneous network processing methods which focus on social recommendation problems, have not considered real-time updating and online incremental processes.

In order to capture the evolution of the recommender systems, Agarwal et al. (2010) proposed a fast online bilinear factor model to learn item-specific factors through online regression by using a large amount of historical data to initialize the online models and thus reducing the dimensionality of the input features. Diaz-Aviles et al. (2012) presented Stream Ranking Matrix Factorization, which utilizes a pairwise approach to matrix factorization in order to optimize the personalized ranking of topics and follows a selective sampling strategy to perform incremental model updates based on active learning principles. Chen et al. (2013) extended the online ranking technique and proposed a temporal recommender system TeRec, through which, users can get recommendations of topics according to their real-time interests and generate fast feedbacks according to the recommendations when posting tweets. Huang et al. (2015) presented a

practical scalable item-based collaborative filtering algorithm, with the characteristics such as robustness to implicit feedback problem. Subbian et al. (2016) proposed a probabilistic neighbourhood-based algorithm for performing recommendations in real-time. The recommendation strategies proposed by Huang et al. (2015) and Subbian et al. (2016) focus on scalability and real-time pruning in recommender system. Our proposed framework considers the combination of the heterogeneous characteristics of social networks and graph-based updating schemes on real-time condition, and thus is substantially different from the above-mentioned systems.

2.2. Distributed Representation Learning

Recently, distributed representation learning has drawn lots of attention due to its effectiveness in representing and extracting useful knowledge in many tasks, including text classification (Tang et al., 2015a), knowledge graph mining (Yang et al., 2015) and recommender system (Wang et al., 2015). Particularly, in text processing field, the word embedding model word2vec developed in Mikolov et al. (2013a,b) makes it possible to train the embedding vectors on large-scale datasets with a single machine. Stochastic Gradient Descent (SGD) is used to train the parameters with two alternative optimized algorithms for speedup, namely hierarchical softmax (Morin and Bengio, 2005) and noise contrastive estimation (Mnih and Teh, 2012). Several usages of distributed representation learning also have been proposed for different applications. For instance, DeepWalk (Perozzi et al., 2014) adapted Skip-Gram (Mikolov et al., 2013b), a widely used language model in natural language processing area, for network representation learning on truncated random walk. LINE (Tang et al., 2015b) is a scalable network embedding algorithm which modelled the first-order and second-order proximities between vertices. Our work is highly built upon these studies. The novelty lies in the idea of adopting the graph embedding methods into the dynamic environment for real-time recommendation.

3. Problem Formulation

In this section, we first introduce the key data structures and the definition of each source for the heterogeneous network. Then, the problem statement of this study is presented. Table 1 summarizes the notations of frequently used variables.

Definition 1. Item Profile *An item is defined as a uniquely post (e.g., a tweet or a news article). In our model, an item can be denoted as a five tuple* $(iId, \mathcal{M}, \mathcal{H}, \mathcal{W}, \rho)$*, representing itemID, named entity, hashtag/category, content, create time respectively.*

Definition 2. User Profile *For each user* u*, we create the user profile as a three tuple* $(uId, \mathcal{L}, \mathcal{D})$*, which indicates userID, user social links and a set of items associated with* u*.*

Table 1: Notations used in the paper.

Symbol	Description
$\mathcal{U}$, $\mathcal{P}$	the set of users and items
$\mathcal{M}$, $\mathcal{H}$, $\mathcal{W}$, $\mathcal{L}$	the set of named entities, hashtags/categories, content words and social links
G_{mix}	heterogeneous user-item (HUI) network
t	the timestamp of heterogeneous user-item network
$\mathbb{R}^d$	d dimensional latent space
$\overrightarrow{v}$, $\overrightarrow{p}$	embeddings of user u and item p, respectively
Δt	the time interval
α, β, γ	model parameters controlling the relative importance of user behavior sequential patterns, social relationships and semantic effects
$\mathcal{R}_i$	the social links of user u_i
P	the transition probability matrix of heterogeneous user-item network
$\widetilde{\mathcal{V}}_t$	the active nodes at timestamp t

Definition 3. User-user Relationship Network *A user-user relationship network can be represented by $G_{uu} = (\mathcal{U}, \varepsilon_{uu})$, where $\mathcal{U} = \{u_1, u_2, ...u_n\}$ is the set of users, and ε_{uu} is the set of edges. Each $e_{ij} \in \varepsilon_{uu}$ is a social link, such as following or friends, between user i and user j.*

Definition 4. Item-item Relationship Network *An item-item relationship network can be represented by $G_{pp} = (\mathcal{P}, \varepsilon_{pp})$, where $\mathcal{P} = \{p_1, p_2, ...p_n\}$ is the set of items, and ε_{pp} denotes the set of edges. If item p_i and item p_j have a semantic link such as Named Entity or Hashtag, there will be an edge $e_{ij} \in \varepsilon_{pp}$ between them, otherwise none.*

Definition 5. User-item Interaction Network *A user-item interaction network can be represented by $G_{up} = (\mathcal{U} \cup \mathcal{P}, \varepsilon_{up})$, where $\mathcal{U} = \{u_1, u_2, ...u_n\}$ is the set of users, $\mathcal{P} = \{p_1, p_2, ...p_n\}$ is the set of items, and ε_{up} denotes the set of edges. If item p_j is of interest to user u_i (based on user activities such as 'clicked', 'retweet', etc), there will be an edge $e_{ij} \in \varepsilon_{up}$ between them, otherwise none.*

Definition 6. Heterogeneous User-Item (HUI) Network *A heterogeneous user-item network can be represented by $G_{mix} = G_{uu} \cup G_{pp} \cup G_{up}$, which consists of the user-user relationship network G_{uu}, the item-item relationship network G_{pp} and the user-item interaction network G_{up}. The same sets of users and items are shared in G_{mix}.*

The heterogeneous user-item network can well capture social relationship influence, semantic effect and user behavior sequential patterns simultaneously. Take the semantic effect as an example, we can interpret it as following: if a user u_i is visiting an item p_j at time slot t and item p_k is more similar with p_j than other items, then u_i is most likely to visit p_k. Our goal is to embed the heterogeneous user-item network into a shared low dimensional space $\mathbb{R}^d$ where d is the dimension. Then, we can get the vector representations of users $\overrightarrow{v}$ and items $\overrightarrow{p}$.

Finally, we formally define the problem investigated in our work. Given a time-stamped heterogeneous user-item network, we aim to provide real-time social recommendations stated as follows.

Problem 1. (Real-time Social Recommendation) *Given a heterogeneous user-item network G_{mix} at timestamp t and a querying user $u \in \mathcal{U}$, the task is to generate a ranked list of user or item recommendations that u would be interested in.*

4. DGE: Dynamic Graph-based Embedding Model

In this section, we propose a novel dynamic graph-based embedding (DGE) model for real-time social recommendation. Firstly, the construction of the HUI network as well as its update process are described in details. Then, we introduce the dynamic graph embedding approach which involves the edge sampling and an incremental learning algorithm. Finally, a list of top-k recommendations can be generated by evaluating the similarities between the learned representations of different vertices.

4.1. Heterogeneous User-Item (HUI) Network

4.1.1. HUI Network Construction

For notational simplicity, we ignore the time-subscript in this subsection. Assume that we are given a set of users $\mathcal{U} = \{u_1, u_2, ...u_m\}$ and a set of items $\mathcal{P} = \{p_1, p_2, ...p_n\}$. To integrate the semantic effects, social relationships and the user behavior sequential patterns simultaneously, we construct a heterogeneous user-item network comprising two types of nodes and three types of edges, as shown in Fig. 2. The two types of nodes which consist of user and item nodes are formed by projecting the user set and item set respectively. The three types of edges are defined as follows:

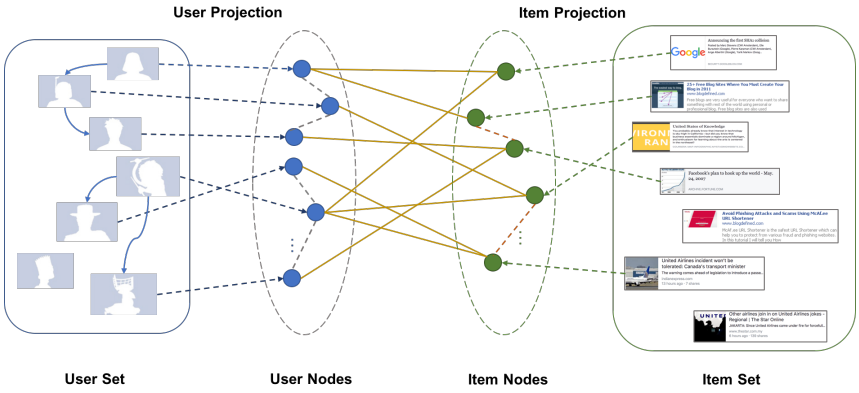


Figure 2: The Heterogeneous User-Item (HUI) Network.

1) Each user node u_i and each item node p_j are connected if user u_i shows an interest on item p_j. In the HUI network, such an edge is indicated by

yellow solid lines. The associated item nodes of the user node u_i are denoted as $\mathcal{I}_p(u_i)$, the associated user nodes of the item node p_j are denoted as $\mathcal{I}_u(p_j)$.

2) Two user nodes u_i and u_j are connected with the property of user similarity $sim_u(u_i, u_j)$ if user u_i and u_j have a social link, such as following or friends. In the HUI network, such edge is indicated by grey dash lines. The adjacent user nodes of the user node u_i are denoted as $\mathcal{A}_u(u_i)$.

3) Two item nodes p_i and p_j are connected with the property of item similarity $sim_p(p_i, p_j)$ if item p_i and p_j have a semantic link such as Named Entity or Hashtag. In the HUI network, such edge is indicated by orange dash lines. The adjacent item nodes of the item node p_i are denoted as $\mathcal{A}_p(p_i)$.

We assume that $\mathcal{R}_i$ is a r-dimensional vector representing the social links of user u_i, where r is the total number of users, and the k-th dimension of vector $\mathcal{R}_i$ equals 1 only if there is an edge between u_i and u_k, otherwise 0. The user similarity $sim_u(u_i, u_j)$ between user u_i and user u_j can be defined as the cosine similarity between the two vectors,

$$sim_u(u_i, u_j) = \frac{\mathcal{R}_i^T \cdot \mathcal{R}_j}{\sqrt{\mathcal{R}_i^T \cdot \mathcal{R}_i} \cdot \sqrt{\mathcal{R}_j^T \cdot \mathcal{R}_j}} \tag{1}$$

Likewise, the item similarity $sim_p(p_i, p_j)$ between two item nodes p_i and p_j is also defined as the cosine similarity between the two corresponding feature vectors, which contain named entity, hashtag/category and the occurrence frequency of words in the item content.

Corresponding to the three types of edges with different characteristics, there are three types of random walk modes, which are between user nodes, between item nodes as well as between user and item nodes. Directly applying random walk to the HUI network does not work due to different edge types, leading to a challenging problem. To this end, we propose a novel way to capture the different edge type characteristic into the transition probability matrix P, where three parameters α, β, γ with $\alpha + \beta + \gamma = 1$ are used to respectively control the relative importance of user behavior sequential patterns, social relationships and semantic effects.

Definition 7. *A transition probability matrix $P \in \mathbb{R}^{(m+n)\times(m+n)}$ is constructed for the HUI network,*

$$P = \begin{pmatrix} P_u & P_{up} \\ P_{pu} & P_p \end{pmatrix} \tag{2}$$

which comprises four matrix blocks $P_u \in \mathbb{R}^{m\times m}$, $P_{up} \in \mathbb{R}^{m\times n}$, $P_{pu} \in \mathbb{R}^{n\times m}$ and $P_p \in \mathbb{R}^{n\times n}$ respectively representing the transition probabilities of random walks between user nodes, from user nodes to item nodes, from item nodes to

user nodes and between item nodes. That is

$$P_{i,j} = Prob(u_j|u_i), \qquad i < m, j < m$$
$$= \begin{cases} 0 & u_j \notin \mathcal{A}_u(u_i) \\ \frac{\beta}{\alpha+\beta} \times \frac{sim_u(u_i,u_j)}{\sum_{u_k \in \mathcal{A}_u(u_i)} sim_u(u_i,u_k)} & u_j \in \mathcal{A}_u(u_i) \end{cases} \tag{3}$$

$$P_{i,m+j} = Prob(p_j|u_i), \qquad i < m, j < n$$
$$= \begin{cases} 0 & p_j \notin \mathcal{I}_p(u_i) \\ \frac{\alpha}{\alpha+\beta} \times \frac{1}{|\mathcal{I}_p(u_i)|} & p_j \in \mathcal{I}_p(u_i) \end{cases} \tag{4}$$

$$P_{m+i,j} = Prob(u_j|p_i), \qquad i < n, j < m$$
$$= \begin{cases} 0 & u_j \notin \mathcal{I}_u(p_i) \\ \frac{\alpha}{\alpha+\gamma} \times \frac{1}{|\mathcal{I}_u(p_i)|} & u_j \in \mathcal{I}_u(p_i) \end{cases} \tag{5}$$

$$P_{m+i,m+j} = Prob(p_j|p_i), \qquad i < n, j < n$$
$$= \begin{cases} 0 & p_j \notin \mathcal{A}_p(p_i) \\ \frac{\gamma}{\alpha+\gamma} \times \frac{sim_p(p_i,p_j)}{\sum_{p_k \in \mathcal{A}_p(p_i)} sim_p(p_i,p_k)} & p_j \in \mathcal{A}_p(p_i) \end{cases} \tag{6}$$

In the above definition, we do not use the same similarity measurement to quantify the user-item connection since the user content and item content adopt independent lexicons and different representation schemes, which means that it is difficult to compute their similarities (e.g., cosine similarity). Besides, selecting different values for parameters α, β and γ corresponds to assign different importance degrees to semantic effects, social relationships and user behavior sequential patterns, which depends on the datasets. In the experiments, we will show that setting the same values for the three parameters, i.e., $\alpha = \beta = \gamma = 1/3$, can lead to the best recommendation results on the two testing datasets.

4.1.2. HUI Network Update

Assume at timestamp t, the current HUI network $G_{mix,t} = (\mathcal{V}_t, \varepsilon_t) = (\mathcal{U}_t, \varepsilon_{uu,t}, \mathcal{P}_t, \varepsilon_{pp,t}, \varepsilon_{up,t})$ contains the user node set $\mathcal{U}_t$, item node set $\mathcal{P}_t$ and their related edge sets $\varepsilon_{uu,t}$, $\varepsilon_{pp,t}$ and $\varepsilon_{up,t}$. Due to the evolving of the network, $\mathcal{U}_t$ and $\mathcal{P}_t$ will contain the sets of the newly attached nodes, denoted as $\Delta\mathcal{U}_t$ and $\Delta\mathcal{P}_t$ respectively, while there exists another subsets of $\mathcal{U}_t$ and $\mathcal{P}_t$ containing the nodes that have changed at the current timestamp, which are denoted as $\Theta\mathcal{U}_t$ and $\Theta\mathcal{P}_t$. Similarly, subsets of $\varepsilon_{uu,t}$, $\varepsilon_{pp,t}$ and $\varepsilon_{up,t}$ contain the newly attached edges, separately denoted as $\Delta\varepsilon_{uu,t}$, $\Delta\varepsilon_{pp,t}$ and $\Delta\varepsilon_{up,t}$, while the subsets of changed edges within $\varepsilon_{uu,t}$, $\varepsilon_{pp,t}$ and $\varepsilon_{up,t}$ at current timestamp are denoted as $\Theta\varepsilon_{uu,t}$, $\Theta\varepsilon_{pp,t}$ and $\Theta\varepsilon_{up,t}$ separately.

It is necessary to update the HUI network from timestamp $t-1$ to timestamp t according to the evolving nodes $(\Delta\mathcal{U}_t \cup \Theta\mathcal{U}_t, \Delta\mathcal{P}_t \cup \Theta\mathcal{P}_t)$ and edges $(\Delta\varepsilon_{uu,t} \cup \Theta\varepsilon_{uu,t}, \Delta\varepsilon_{pp,t} \cup \Theta\varepsilon_{pp,t}, \Delta\varepsilon_{up,t} \cup \Theta\varepsilon_{pp,t})$. This can be easily achieved by updating the two types of nodes and three types of edges in HUI network. For instance, u new user nodes and e new user-user edges are added to the HUI network and their similarities of user social links are computed among related nodes. Accordingly, the transition probability matrix P can be easily updated.

The active nodes at timestamp t (denoted as $\widetilde{\mathcal{V}}_t$) are defined as the union of the evolving nodes $(\Delta\mathcal{U}_t \cup \Theta\mathcal{U}_t, \Delta\mathcal{P}_t \cup \Theta\mathcal{P}_t)$ and the nodes incident upon the evolving edges $(\Delta\varepsilon_{uu,t} \cup \Theta\varepsilon_{uu,t}, \Delta\varepsilon_{pp,t} \cup \Theta\varepsilon_{pp,t}, \Delta\varepsilon_{up,t} \cup \Theta\varepsilon_{pp,t})$. That is

$$\begin{aligned}\widetilde{\mathcal{V}}_t = \Delta\mathcal{U}_t \cup \Theta\mathcal{U}_t \cup \Delta\mathcal{P}_t \cup \Theta\mathcal{P}_t \cup \{u_i | \exists e_u \in \Delta\varepsilon_{uu,t} \cup \Theta\varepsilon_{uu,t}, e_u = (u_i, u_j)\} \\ \cup \{p_i | \exists e_p \in \Delta\varepsilon_{pp,t} \cup \Theta\varepsilon_{pp,t}, e_p = (p_i, p_j)\} \\ \cup \{u_k, p_f | \exists e_{up} \in \Delta\varepsilon_{up,t} \cup \Theta\varepsilon_{up,t}, e_{up} = (u_k, p_f)\}\}\end{aligned} \tag{7}$$

The underlying principle of the network constructing and updating process can be analogous to the case of adopting sliding window schema to manage continuous data streams. The construction process of HUI network is based on the historical records, and the updating course of the network can be conducted only within several timestamps like a certain length sliding window. The worst case happens only when all nodes $\{v_i | v_i \in \mathcal{V}_t\}$ have changed within timestamp t. In such case, the retraining process of the whole HUI network is inevitable.

4.2. Heterogeneous User-Item Network Embedding

Inspired by DeepWalk (Perozzi et al., 2014) and the idea of modelling document (Djuric et al., 2015; Le and Mikolov, 2014) in natural language processing, our model contains two main stages, heterogeneous random walk and model learning process. In this section, we will illustrate each stage in details.

4.2.1. Heterogeneous Random Walk

According to the previous work (Jeh and Widom, 2003), random walk can be used to define proximity, but it is only limited to the network with one type of nodes and links. In order to extend random walk into heterogeneous networks with multiple nodes and various types of edges, the transition probability matrix P defined in Section 4.1 is introduced to treat different kinds of nodes and edges equally.

Given the length of random walk as h and the total number of random walks as l, the starting step will be performed at each of the active node $\widetilde{\mathcal{V}}_t$ at timestamp t. Based on the updated transition probability matrix P, the heterogeneous random walk will generate possible route sequesces for active nodes, denoted as $S = \{s_1, s_2, ..., s_{|\widetilde{\mathcal{V}}_t|}\}$. The detailed procedure is proceeded as follows.

1. When the walker is in the user node u_i, it will jump to either one of its associated item nodes $p_j \in \mathcal{I}_p(u_i)$ or one of its adjacent user nodes $u_j \in \mathcal{A}_u(u_i)$, with probabilities accessed from the transition probability matrix P.

2. When the walker is in the item node p_i, it will jump to either one of its associated user nodes $u_j \in \mathcal{I}_u(p_i)$ or one of its adjacent item nodes $p_j \in \mathcal{A}_p(p_i)$, with probabilities accessed from the transition probability matrix P.

Such hop process is repeated until finishing h hops, which is taken as a single random walk. And since the total number of l random walks are performed, the whole procedure generates $l \times h$ hops. The encountered combination of nodes for node v_i during these hops is denoted as the possible route sequence of v_i. In random walk, the jump cannot go directly back to the previous node, for example, v_i to v_j back to v_i is not allowed, which in order to avoid getting stuck in some hops with high probabilities in P. Algorithm 1 summerizes the procedure of the heterogeneous random walk for the active nodes at each timestamp.

Algorithm 1: Heterogeneous Random Walk

Input: Transition probability P, active node set $\widetilde{V}_t$, number of random walks l, length of random walk h.
Output: The set S of possible route sequence for each node in active node set $\widetilde{V}_t$

```
for ∀v_i ∈ Ṽ_t do
    for i = 1 to l do
        Perform an h-hop random walk starting at v_i using the transition
        probability matrix P
    end
    Possibile route sequence s_i for node v_i
end
```

4.2.2. Incremental Network Embedding Learning

During the model learning process, the heterogeneous random walk will be performed on the initial HUI network $G_{mix} = (\mathcal{V}, \varepsilon)$ firstly, and it results in a set of possible route sequences $S = \{s_1, s_2, ..., s_{|\widetilde{\mathcal{V}}_t|}\}$, where each sequence can be denoted as $s = \{v_1, v_2, ..., v_{|s|}\}$. DeepWalk treats each route sequence s as a word sequence by regarding nodes as words. Then by introducing Skip-Gram, a widely used word representation learning algorithm, DeepWalk is able to learn node representations from the sequence set S. Similarly, our model also adopts Skip-Gram to learn the representation of each node. More specifically, when given a node route sequence $s = \{v_1, v_2, ..., v_{|s|}\}$, each node v_i has $\{v_{i-T}, ..., v_{i+T}\} \backslash \{v_i\}$, as its local context nodes. Thus, DGE model learns node representations by maximizing the average log probability of predicting context nodes:

$$\mathcal{L}(s) = \frac{1}{|s|} \sum_{i=1}^{|s|} \sum_{i-|T| \leq j \leq i+|T|} logPr(v_j|v_i) \tag{8}$$

where v_j is the context node of the node v_i, and the probability $Pr(v_j|v_i)$ is defined using the softmax function:

$$Pr(v_j|v_i) = \frac{exp(\boldsymbol{v}'_j \cdot \boldsymbol{v}_i)}{\sum_{v \in \mathcal{V}} exp(\boldsymbol{v}' \cdot \boldsymbol{v}_i)} \tag{9}$$

where $\boldsymbol{v}_i$ is the representation of the center node v_i and $\boldsymbol{v}'_j$ is the context representation of its context node v_j. Then subsequently, during incremental learning process at each timestamp $t > 1$, the heterogeneous random walk procedure and Skip-Gram will be proceeded on active node set $\widetilde{\mathcal{V}}_t$ and their related edges.

Given that calculating Eq. (9) directly is not feasible and will lead expensive computing cost in practical implementation. Therefore, a computational efficient approximation of the full softmax called hierarchical softmax (Mikolov et al., 2013b), is introduced to solve this problem. The hierarchical softmax uses a binary tree representation for every context node $v_j \in \mathcal{V}$ as its leaves, and each tree node is explicitly associated with an embedding vector θ for computing the relative probability to take the branch. Each leave can be reached by an appropriate path from the root of the tree. In this way, instead of evaluating all the $|\mathcal{V}|$ nodes, it needs to evaluate only about $log(|\mathcal{V}|)$ nodes to obtain the probability distribution.

More precisely, given the representation $\boldsymbol{v}_i$ of node v_i for target context v_j, let $L(v_j)$ be the length of its corresponding path, and let $b_n^{v_j} = 0$ when the path to v_j takes the left branch at the n-th layer and $b_n^{v_j} = 1$ otherwise. Then, the hierarchical softmax defines $Pr(v_j|v_i)$ as follows:

$$Pr(v_j|v_i) = \prod_{n=2}^{L(v_j)} ([\sigma(\boldsymbol{v}_i^T \theta_{n-1}^{v_j})]^{1-b_n^{v_j}} \cdot [1 - \sigma(\boldsymbol{v}_i^T \theta_{n-1}^{v_j})]^{b_n^{v_j}}) \tag{10}$$

where $\sigma(z) = \frac{1}{1+exp(-z)}$. All parameters are trained by using the Stochastic Gradient Descent method. During the training, the algorithm iterates over the nodes through all possible route sequences, and at each time, a target node v_j with its context window is used for update. After computing the hierarchical softmax according to Eq. (10), the error gradient is obtained via backpropagation and we use the gradient to update the parameters in our model. To derive how θ is updated at each time step, the gradient for $\theta_{n-1}^{v_j}$ is computed as follows:

$$\frac{\partial \mathcal{L}(v_j, n)}{\partial \theta_{n-1}^{v_j}} = [1 - b_n^{v_j} - \sigma(\boldsymbol{v}_i^T \theta_{n-1}^{v_j})]\boldsymbol{v}_i \tag{11}$$

In this way, $\theta_{n-1}^{v_j}$ can be updated as:

$$\theta_{n-1}^{v_j} \leftarrow \theta_{n-1}^{v_j} + \eta[1 - b_n^{v_j} - \sigma(\boldsymbol{v}_i^T \theta_{n-1}^{v_j})]\boldsymbol{v}_i \tag{12}$$

where η denotes the learning rate. To derive how the representation of the center node is updated, the fradient for $\boldsymbol{v}_i$ is computed as follows:

$$\frac{\partial \mathcal{L}(v_j, n)}{\partial \boldsymbol{v}_i} = [1 - b_n^{v_j} - \sigma(\boldsymbol{v}_i^T \theta_{n-1}^{v_j})]\theta_{n-1}^{v_j} \tag{13}$$

With this derivative, an embedding vector $\boldsymbol{v}_i$ in the context of node v_j can be updated as follows:

$$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \eta \sum_{n=2}^{L(v_j)} \frac{\partial \mathcal{L}(v_j, n)}{\partial \boldsymbol{v}_i} \tag{14}$$

In Algorithm 2, we summerize the learning process using hierarchical softmax for proposed DGE model. The algorithm iterates through all possible route sequences and updates the embedding vectors until the procedure converges. In each iteration, given a current node, the algorithm first obtains its embedding vectors and computes its context embedding vector. Based on the derivative above, the binary tree in hierarchical sampling is updated followed by the embedding vector. Given the vector size of d, the leaf nodes number $|V|$, the sequence length $|s|$ within one iteration and window length $|T|$, then the time complexity for an iteration is $\mathcal{O}(d \cdot |T| \cdot |s| \cdot log(|V|))$.

Algorithm 2: Heterogeneous Softmax Algorithm for Learning Parameters of DGE

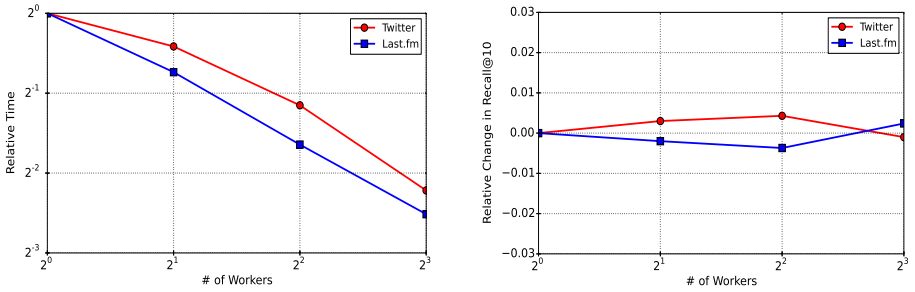
Input: Possible route sequence set S, window length $|T|$, embedding vector dimension d, sequence length $|s|$.
Output: The embedding representation $\boldsymbol{v}_i$ of node v_i

Initialize the parameters randomly;
Shuffle the dataset;
repeat
  Sample a route sequence $s = \{v_1, v_2, ..., v_{|s|}\}$ from S;
  for $i = 1$ *to* $|s|$ **do**
    Set $e \leftarrow 0$;
    Compute the representation $\boldsymbol{v}_i$ of v_i;
    for *each* $v_j \in s[i - |T|, i + |T|]$ **do**
      for n = 2 *to* $L(v_j)$ **do**
        $q \leftarrow \sigma(\boldsymbol{v}_i \cdot \theta_{n-1}^{v_j})$;
        $g \leftarrow \eta \cdot (b_n^{v_j} - 1 - q)$;
        $e \leftarrow e + g \cdot \theta_{n-1}^{v_j}$;
        Update $\theta_{n-1}^{v_j} \leftarrow \theta_{n-1}^{v_j} + g \cdot \boldsymbol{v}_i$;
      end
      Update $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \eta \cdot e$;
    end
  end
until *convergence*;

4.2.3. Parallelizability

For real-world social networks, the frequency distribution of vertices in random walks follows a power law which results in a long tail of infrequent vertices (Perozzi et al., 2014). Therefore, the updates of vertices' representation will

be sparse in nature. Based on this, we adopt the lock-free solutions in the work (Recht et al., 2011) to parallelize asynchronous stochastic gradient descent (ASGD). Given that our updates are sparse and we do not acquire a lock to access the model shared parameters, ASGD will achieve an optimal rate of convergence. Fig. 3 presents the effects of parallelizing DGE model with multiple threads. It shows the speed up in processing Twitter and Last.fm datasets is consistent as we increase the number of workers to 8 (Fig. 3a). It also shows that there is no loss of predictive performance relative to the running DGE serially (Fig. 3b).



(a) Running Time (b) Performance

Figure 3: Effects of parallelizing DGE model.

4.3. Recommendation Using DGE

Once we have learnt the model parameters, recommendations can be made by utilizing the embeddings for each vertex in the social network. In this section, we propose the top-K recommendation algorithms for a user to select potential friends and interested items respectively.

4.3.1. Recommending top-K friends:

This task is to recommend top-K friends that a user u would like to follow in the social network. More precisely, Given a target user $u_i \in \mathcal{U}$ with the query time t, for each user node u_j who has not been connected with u_i, we compute its ranking score as in Eq. (15), and then select the k ones with the highest ranking scores as recommendations.

$$S(u_i, u_j, t) = \sum_{k=1}^{D} x_{ik} \cdot y_{jk} \tag{15}$$

where $u_i = (x_{i1}, x_{i2}, ..., x_{iD})$, $u_j = (y_{j1}, y_{j2}, ..., y_{jD})$, D is the dimension of the representation vector.

The straightforward method of generating the top-k friends needs to compute the ranking scores for almost all users according to Eq. (15), which is computationally inefficient, especially when the number of users becomes large. To

speed up the process of producing recommendations, we extend the Threshold-based Algorithm (TA) (Fagin et al., 2003), which is capable of finding the top-k results by examining the minimum number of users.

We first pre-compute the ordered lists of users, where each list corresponds to a dimension of the user's representation vector $u_w = (y_{w1}, y_{w2}, ..., y_{wD})$. So, we could get D lists of sorted users, $L_n, n \in \{1, 2, ..., D\}$, where users in each list L_n are sorted according to y_{wn}. Given a query $q = (u_i, t)$, we run Algorithm 3 to compute the top-k users from the D sorted lists and return them in the priority list L. As shown in Algorithm 3, we first maintain a priority list PL for the D lists where the priority of a list L_n is the ranking score $S(u_i, u_w, t)$ of the first user w in L_n (Lines 2-6). In each iteration, we select the most promising user (i.e., the first user) from the list that has the highest priority in PL and add it to the resulting list L (Lines 9-16). When the size of L is no less than k, we will examine the k-th user in the resulting list L. If the ranking score of the k-th user is higher than the threshold score T_s, the algorithm terminates early without checking any subsequent users (Lines 18-20). Otherwise, the k-th user $w^{'}$ in L is replaced by the current user w if w's ranking score is higher than that of $w^{'}$ (Lines 21-24). At the end of each iteration, we update the priority of the current list as well as the threshold score (lines 27-32).

Eq. (16) illustrates the computation of the threshold score T_s, which is obtained by aggregating the maximum y_{wn} represented by the first user in each list L_n. Consequently, it is the maximum possible ranking score that can be achieved by the remaining unexamined items. Hence, if the ranking score of the k-th user in the resulting list L is higher than the threshold score, L can be returned immediately because no remaining user will have a higher ranking score than the k-th user.

$$T_s = \sum_{n=1}^{D} x_{in} \cdot \max_{w \in L_n} y_{wn} \tag{16}$$

4.3.2. Recommending top-K items:

This task is to recommend top-K items that a user u would like to be interested in. From the fresh-based perspective, the most recent items that a user shows an interest on could better reflect his/her current preference and they should contribute more in the computation of the recommendations (Stefanidis et al., 2013). Thus, we use the exponential function $f(t_1, k) = e^{-k(t-t_1)}$, where t_1 is the timestamp of item and k is employed to adjust the decay rate, to reflect the freshness of items. According to this, the ranking score of recommendations can be computed as follows:

$$S(u_i, p_j, t) = f(t_j, k) \sum_{n=1}^{D} x_{in} \cdot z_{jn} \tag{17}$$

where $u_i = (x_{i1}, x_{i2}, ..., x_{iD})$ is the representation of target user and $p_j = (z_{j1}, z_{j2}, ..., z_{jD})$ is the representation of a candidate item. Once the newly arrived items have been settled in ordered candidate item lists, as time goes on,

Algorithm 3: Threshold-based algorithm

Input: A query $q = (u_i, t)$, ranked lists $(L_1, ..., L_D)$.
Output: List L with all the k highest ranked users.

Initialize priority lists PL, L and the threshold score T_s;
for $n = 1$ *to* D **do**
    $w = L_n.getfirst()$;
    Compute $S(u_i, u_w, t)$ according to Eq. (15);
    $PL.insert(n, S(u_i, u_w, t))$;
end
Compute T_s according to Eq. (16);
while *true* **do**
    $nextListToCheck = PL.getfirst()$;
    $PL.removefirst()$;
    $w = L_{nextListToCheck}.getfirst()$;
    $L_{nextListToCheck}.removefirst()$;
    if $w \notin L$ **then**
        if $L.size() < k$ **then**
            $L.insert(w, S(u_i, u_w, t))$;
        else
            $w' = L.get(k)$;
            if $S(u_i, u'_w, t) > T_s$ **then**
                break;
            end
            if $S(u_i, u'_w, t) < S(u_i, u_w, t)$ **then**
                $L.remove(k)$;
                $L.insert(w, S(u_i, u_w, t))$;
            end
        end
    end
    if $L_{nextListToCheck}.hasMore()$ **then**
        $w = L_{nextListToCheck}.getfirst()$;
        Compute $S(u_i, u_w, t)$ according to Eq. (15);
        $PL.insert(nextListToCheck, S(u_i, u_w, t))$;
        Compute T_s according to Eq. (16);
    else
        break;
    end
end

the decay value of all items freshness will be the same as $e^{-k \cdot \Delta t}$, Δt is the time interval, without influencing the order of them. Thus this allows us to leverage TA algorithm for retrieving and recommending items the same as friends recommendation.

4.4. Framework Extensibility

Here we discuss the extendability of our proposed framework, which we believe may be of interest.

4.4.1. Multiple Social Networks

Intuitively, our DGE model could be extended to multiple sources for a user who is affiliated to them. For instance, if a user has an account in Facebook and also in Twitter, then both kinds of social sources can bring valuable and multiple information which could assist to improve the recommendation performance. Nowadays, some approaches have been designed to apply the recommendation strategies to support users operating in multiple social sites (De Meo et al., 2011). Other researches concentrate on the construction of a global user profile with the integration of multiple information from different sources (Buccafurri et al., 2016). Zhang and Yu (2016) proposed an unsupervised network alignment framework (UNICOAT) to address the partial co-alignment problem and discover the potential links between users and between locations for multiple sources. In Jia et al. (2016), the authors introduced a deep learning-based approach integrating fusing social networks to predict volunteerism tendency. More specifically, it learns predictive models independently for multiple sources with the input of the hyper representations of features extracted from multiple sites separately, and then weighted sum of these models into the final predictive model.

Inspired by Jia et al. (2016), we can build our heterogeneous network separately for multiple sources, and learn the node representations separately. Then, a weighted average can be used to form the final representation of each node. Specifically, let $\boldsymbol{v}_i^k$ be the learned representation of user/item node v_i in the k-th source, and thus the final representation of node $\boldsymbol{v}_i$ can be derived as

$$\boldsymbol{v}_i = \sum_{k \in K} \lambda^k \boldsymbol{v}_i^k \tag{18}$$

where K represents the source set, and λ^k represents the weight of different sources in k. Thus the objective function of Eq. (8) can be redefined as

$$\mathcal{L}(s) = \frac{1}{|s|} \sum_{i=1}^{|s|} \sum_{i-|T| \leq j \leq i+|T|} logPr(v_j|v_i) + \eta R \tag{19}$$

where R is the regularization term defined as

$$R = -\sum_{i \in |S|} \sum_{k=1}^{K} \lambda_i^k ||\boldsymbol{v}_i^k - \boldsymbol{v}_i||_2^2 \tag{20}$$

and η is a parameter used to control the weight of the regularization term. By maximizing this objective function, different multiple sources can be collaborated to learn the robust node representations.

4.4.2. User Bias

In some practical scenarios where users rate for the related items, the matrix blocks P_{up} and P_{pu} in the transition probability matrix can be defined incorporat- ing user bias:

$$\begin{aligned} P_{i,m+j} &= Prob(p_j|u_i), \qquad i < m, j < n \\ &= \begin{cases} 0 & p_j \notin \mathcal{I}_p(u_i) \\ \frac{\alpha}{\alpha+\beta} \times \frac{w_{ij}}{\sum_{p_k \in \mathcal{I}_p(u_i)} w_{ik}} & p_j \in \mathcal{I}_p(u_i) \end{cases} \end{aligned} \tag{21}$$

$$\begin{aligned} P_{m+i,j} &= Prob(u_j|p_i), \qquad i < n, j < m \\ &= \begin{cases} 0 & u_j \notin \mathcal{I}_u(p_i) \\ \frac{\alpha}{\alpha+\gamma} \times \frac{w_{ji}}{\sum_{u_k \in \mathcal{I}_u(p_i)} w_{ki}} & u_j \in \mathcal{I}_u(p_i) \end{cases} \end{aligned} \tag{22}$$

where w_{ij} denotes the rating score that the user u_i assigns to item p_j. Then the same learning procedures (Section 4.2) can be used to achieve the representations of the nodes.

5. Experimental Setup

5.1. Dataset Description

For experimental study, we evaluate the proposed DGE model on two kinds of real-world datasets: Twitter and Last.fm. We downloaded the Twitter dataset from Twitter API[2], which includes users and their posts. We collected the Last.fm dataset through Last.fm API[3], which contains users and artists. The statistics of each dataset is summerized in Table 2. For both datasets, the user-user links are constructed from bi-directional friendships between social network users, user-item links are constructed from the user listening or posting behaviour, and item-item link are constructed if the two artists share the same tag or the two posts have the same hashtag.

Table 2: Some statistics of the datasets.

Dataset	**user**	**item**	**user-user links**	**user-item links**	**item-item links**
Twitter	87,287	7,855,830	537,251	86,414,130	38,493,509
Last.fm	50,000	11,215	291,805	14,358,010	2,264,562

[2]https://dev.twitter.com/docs, accessed: March 15, 2017.
[3]http://www.last.fm/api/, accessed: March 15, 2017.

5.2. Comparative Approaches

We compared the proposed approach with four state-of-the-art methods:

- **Popular in Neighborhood (PN).** Given a personalized graph G_u, the Popular in Neighbourhood (PN) algorithm ranks the candidate items based on the number of actions all users performed on them within the context of G_u. Hence, the algorithm ranks the candidate entities that appear in G_u based on how popular they are.

- **Weighted Regularized Matrix Factorization (WRMF).** This is a state-of-the-art offline matrix factorization model for item prediction introduced by Hu et al. (2008). Their method outperforms neighbourhood based (item-item) models in the task of item prediction for implicit feedback datasets. The model is computed in batch mode, assuming that the whole stream is stored and available for training.

- **Stream Ranking Matrix Factorization (RMFX).** It is proposed in Ernesto's recent work (Diaz-Aviles et al., 2012), which can achieve partly online and much quicker updates of matrix factorization for item prediction.

- **DeepWalk** (Perozzi et al., 2014). It uses local information obtained from truncated random walks to learn latent representations of nodes in a graph. It is an extended application of word2vec-based model.

- **LINE-2nd**. We also adopt the LINE (Tang et al., 2015b) second-order (2nd) version in order to make the comparison to our proposed context embedding model. According to Xie et al. (2016), the author builds multiple graphs incorporating geographical influence, temporal cyclic effect and semantic effect. Similarly, in this paper, we build multiple graphs integrating user relationships, user-item interactions and semantic influence into one model.

WRMF setup is as follows: $\lambda_{WRMF} = 0.015$, $C = 1$, $epochs = 15$, which corresponds to a regularization parameter, a confidence weight that is put on positive observations, and to the number of passes over observed data, respectively. For RMFX, we set regularization constants $\lambda_{RMFX} = 0.1$, learning rate $\eta_0 = 0.1$, and a learning rate schedule $\alpha = 1$, and find that the setting gives good performance. Moreover, the number of iterations is set to the size of the reservoir. For all the embedding algorithms (DeepWalk, LINE, and our model), the embedding dimensionality is set to 128. We tried dimensionalities in the range [16, 256] and found that 128 generally gives the best results. Context window length is set to 8, walk length is set to 40, walks per vertex is set to 30.

5.3. Evaluation Criteria

Given a dataset $\mathcal{D}$ which includes user profile and item profile, we first rank them according to their tweets timestamp in Twitter dataset or listening

timestamp in Last.fm dataset. Then we use the 80-th percentile as the cut-off point so that user-item interaction behaviors before this point will be used for training and the rest are for testing. In the training dataset, we choose the last 10% records as the validation data to tune the model hyper-parameters such as the dimension of the latent space. According to the above dividing strategies, we split the dataset $\mathcal{D}$ into the training set $\mathcal{D}_{train}$ and the test set $\mathcal{D}_{test}$.

Since we are interested in measuring top-k recommendation instead of rating prediction, we measure the quality by looking at the *Recall@K* metric, which is widely used for evaluating top-k recommender systems (Deshpande and Karypis, 2004; Cremonesi et al., 2010). We show the performance when $k = \{1, 5, 10\}$, as a greater value of k is usually ignored for a typical top-k recommendation task (Cremonesi et al., 2010).

In our recommender system setting, the recall metric is defined as follows:

- *Recall@K* (also known as *hit rate*) is the proportion of relevant items/users found in the top-k recommendations. A larger recall value indicates that the system is able to recommend more satisfactory items or users, leading to a better performance. Formally, we define *hit@k* for a single test case as either the value 1, if the test item or user appears in the top-k results, or the value 0, if otherwise. The overall Recall@k is computed by averaging over all test cases:

$$Recall@K := \frac{\#hit@K}{|\mathcal{D}_{test}|} \tag{23}$$

  where $\#hit@K$ denotes the number of hits in the whole test set.

Another evaluation metric we used is average reciprocal hit-rank (ARHR) (Deshpande and Karypis, 2004) which in our setting we define as follows:

- *ARHR* (average reciprocal hit-rank) is a weighted version of *hit rate* that rewards each hit based on where it occurs in the top-k list. If we take top-k item recommendation as an example, for a target user u, let h be the number of the true interested items in the recommended top-k list, and n_u be the number of u's true interested items in the test dataset. The ARHR is to measure the effectiveness of ranking for each target user. When $p_1, p_2, ..., p_h$ are the positions of the true interested items in the recommended top-k items, the ARHR for u can be defined as

$$ARHR := \frac{1}{n_u} \cdot \sum_{i=1}^{h} \frac{1}{p_i} \tag{24}$$

  As the true interested items appear with high ranks in the recommended items, this measure becomes larger.

In top-k friend recommendations, let h represents the number of the true friends in the recommended top-k list, and n_u represents the number of u's true friends in the test dataset, for each target user u, the average reciprocal hit-rank can be computed similarly.

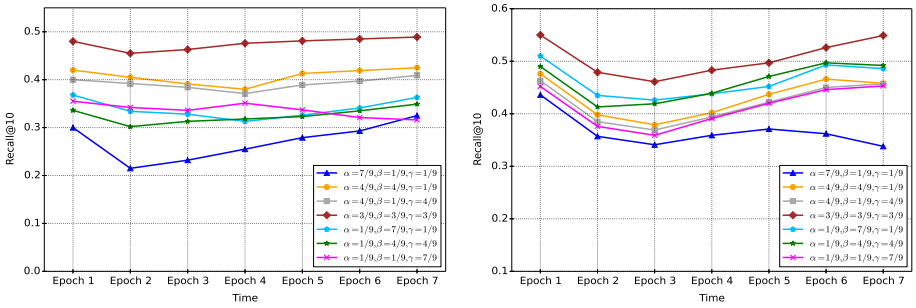
6. Experimental Results

6.1. Sentivity to Parameters

In this section, we first analyze the performance sensitivity to the three trade-off parameters, which control the relative importance of user behavior sequential patterns, social relationships and semantic effects respectively. Additionally, the performance sensitivity to the random walk parameters is also analyzed.

6.1.1. Trade-off Parameters

To investigate how the performance of the DGE model is effected by the relative importance of user behaviour sequential patterns, social relationship and semantic effect, we run the method using various trade-off parameters. The performance is evaluated in terms of the *Recall*@10 value obtained at each timestamp. Fig. 4 plots the *Recall* score on the two datasets using different trade-off parameters.



(a) Twitter (b) Last.fm

Figure 4: Sensitivity to trade-off parameters.

From Fig. 4, we can see that, on the two testing datasets, the worst results are obtained when the relative importance of user behaviour sequential patterns is set very large while the relative importance of social relationship and semantic effects are set very small, i.e. $\alpha = 7/9$, $\beta = 1/9$, $\gamma = 1/9$. This is because in the heterogeneous social network, the content information of items and the influence among user-relationships encoded in the edges is the essential motivation to attract the user to click related items. Especially, in Last.fm dataset, this kind of settings causes a noticeable drop of the *Recall* value at the last three timestamps. The main reason may be that during that time period the users constructed friendship more widely than before.

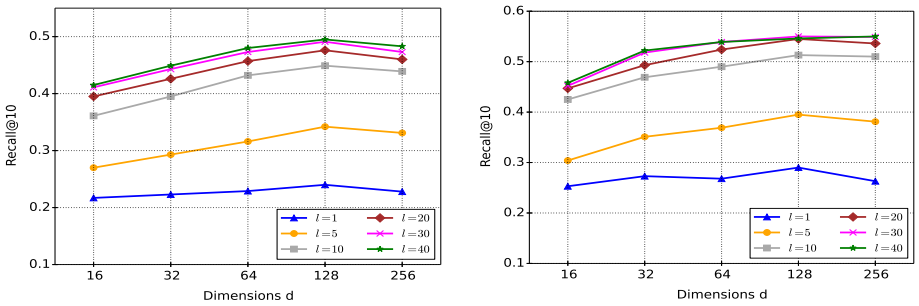
On the other hand, when setting the relative importance of user behaviour sequence patterns, social relationships and semantic effects to the same level as $\alpha = 3/9$, $\beta = 3/9$ and $\gamma = 3/9$, the best *Recall* value is obtained with at least 0.1 improvement, which is very significant. Therefore, in all experiments, except stated otherwise, the relative importance of these three aspects are set to the

same level. The performance sensitivity to the trade-off parameters provides a strong evidence of integrating multiple information in the recommender system.

6.1.2. Effect of Dimensionality and Sampling Frequency

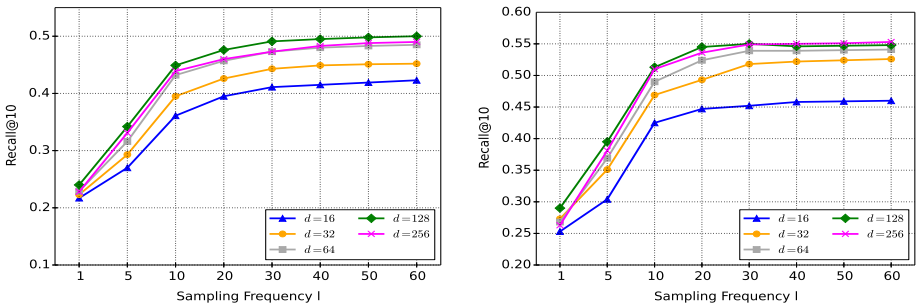
Tuning model parameters is critical to the performance of the proposed model. In this experiment, we study the influence of the embedding dimension d and the number of samples l by fixing the window size $|T| = 8$ and the random walk length $h = 40$. We then vary the number of dimensions d and number of walks started per node l to determine their impact on the recommendation performance. The results are shown in Fig. 5.

From Fig. 5, similar observations can be made on both datasets. It can be observed that recommendation *Recall* value of DEG model is not highly sensitive to the dimension d, but still presents a tendency that its recommendation accuracy increases with the increasing number of dimension d holistically, and then it reaches peak when d is around 128. However, DGE is sensitive to the number of samples l, the *Recall* score varies a lot. First, the performance of DEG increases quickly with the increasing number of l, this is because the model has not achieved convergence. Then, it does not change significantly when the number of samples becomes large enough, since the model DGE has converged. Thus, to achieved a satisfying trade off between effectiveness and efficiency of model training, we set $l = 30$ and $d = 128$ on both datasets.



(a1) Twitter Dataset (a2) Last.fm Dataset

(a) Stability over dimensions, d



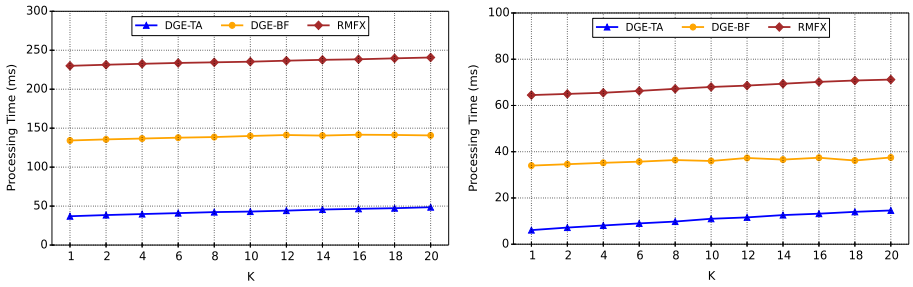
(b1) Twitter Dataset (b2) Last.fm Dataset

(b) Stability over number of samples, l

Figure 5: Effect of Dimensionality and Sampling Frequency.

6.2. Online Recommendation Efficiency

In this section, we evaluate the online recommendation efficiency with Twitter and Last.fm datasets. We test the average time cost of top-k recommendations on two methods, DGE-TA and DGE-BF which utilize the knowledge learnt by DGE to produce recommendations. DGE-TA uses the proposed TA-based query processing technology described in Section 4.3 to produce top-k recommendation results. In DGE-BF, we adopt a brute-based algorithm by scanning all recommendation candidates and computing their ranking scores, to produce top-k recommendations with k highest ranking scores. In addition, we also use the state-of-the-art online recommendation algorithm RMFX to produce top-k recommendations because the other baselines are offline methods and they need to retrain the model during each updating period before recommendation task. Thus here we mainly focus on the online algorithms efficiency comparison.



(a) Twitter dataset (b) Last.fm dataset

Figure 6: Online recommendation efficiency.

Fig. 6 presents the results of different methods with varying number of k from 1 to 20 for Twitter and Last.fm datasets. On average for k equals to 10, our DGE-TA produces top-10 recommendations for Twitter dataset in 43 ms, and for Last.fm dataset in 11 ms. From the figures we conclude several observations that: 1) DGE-TA outperforms the other two methods significantly in both datasets, which verify that the benefits gained by proposed TA-based query processing techinique; 2) DGE-BF ranks the second in top-k recommendation efficiency in both datasets for RMFX generates ranking scores using large amount of matrix operations while DGE uses inner product of two vectors as shown in Eq. (15) and (17); 3) the time cost of DGE-TA method grows with the increasing number of k for DGE-TA needs to scan more recommendation candidates to find top-k recommendations, but DGE-TA is still much efficient than the other two recommendation algorithms since the value of k is normally constrained in a small range; 4) The time cost of each algorithm in Twitter is more expensive than that in Last.fm, showing that if a dataset contains more data, it requires more processing time to produce top-k recommendations.

6.3. Recommendation Effectiveness

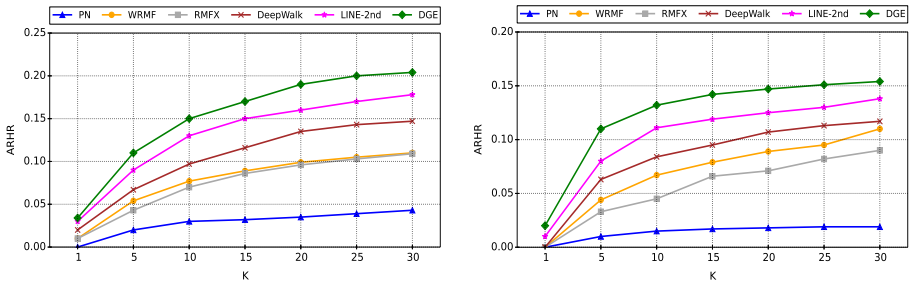
Table 3: Top-k items recommendation effectiveness.

Methods	Twitter			Last.fm		
	Recall@1	Recall@5	Recall@10	Recall@1	Recall@5	Recall@10
PN	0.068	0.109	0.189	0.143	0.203	0.261
WRMF	0.156	0.235	0.314	0.227	0.293	0.385
RMFX	0.125	0.201	0.283	0.196	0.278	0.355
DeepWalk	0.194	0.278	0.351	0.265	0.341	0.423
LINE-2nd	0.223	0.297	0.382	0.271	0.357	0.441
DGE	**0.315**	**0.397**	**0.481**	**0.392**	**0.462**	**0.552**

Table 4: Top-k friends recommendation effectiveness.

Methods	Twitter			Last.fm		
	Recall@1	Recall@5	Recall@10	Recall@1	Recall@5	Recall@10
PN	0.052	0.085	0.161	0.107	0.157	0.223
WRMF	0.115	0.188	0.275	0.176	0.256	0.329
RMFX	0.104	0.158	0.219	0.139	0.232	0.304
DeepWalk	0.149	0.221	0.310	0.216	0.289	0.367
LINE-2nd	0.176	0.232	0.329	0.216	0.311	0.385
DGE	**0.246**	**0.303**	**0.385**	**0.293**	**0.352**	**0.409**

Table 3 and 4 summerize the item and friend recommendation performance for the state-of-the-art methods and the DGE model. Generally speaking, it can be shown from these two tables that the *Recall*@K value grows gradually along with the increasing number of K, and the performance of item recommendation is better than friend recommendation. Besides, we can also observe on both datasets that: Firstly, embedding-based algorithms (DeepWalk, LINE-2nd and DGE) consistently perform better than non-embedding based benchmarks (PN, WRMF and RMFX). For instance, if we consider item recommendation with *Recall*@10 in the Twitter dataset, as shown in Table 3, DeepWalk correctly predicts 35.1% of items, while the best performance of non-embedding based algorithms WRMF correctly predicts 31.4% of items. It is because embedding-based algorithms can fully explore the network structure of the given information, which alleviates the issues of sparse and noisy signals. Secondly, among embedding-based algorithms, DeepWalk is only applicable to homogeneous networks, while LINE-2nd and DGE are capable of handling heterogeneous networks, and thus LINE-2nd and DGE perform better than DeepWalk algorithm. Besides, through considering the semantic effects, social relationships and user behaviour sequential patterns as well as their potential relations simultaneously, DGE encapsulates more contextual information, leading to more informative updates and robustness. Therefore, DGE performs better than LINE-2nd method. The incremental training ability makes DGE update process more efficiently and timely, which also contributes to the better performance of the model.



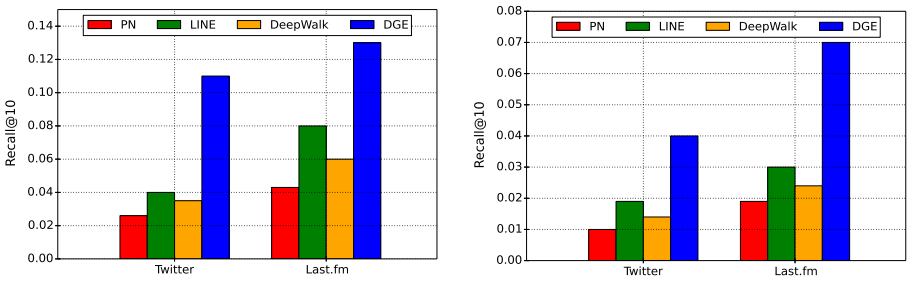
(a) Top-k items recommendation (b) Top-k friends recommendation

Figure 7: Recommendation performance with regard to ARHR.

Fig. 7 compares the performance of alternative approaches taking average reciprocal hit-rank (ARHR) as metric. During experiments, we vary the number of recommendations K from 1 to 30. As expected, our DGE model performs better with ARHR as well, and LINE-2nd ranks the second place, which shows the same orders in Table 3 and 4. As can be seen from Fig. 7a, when we recommend more items, since we have more chances to answer the true interested items correctly, ARHR grows gradually with increasing number K. The same trends appeared in the friend recommendation task.

6.4. Cold Start Problem

In this experiment, we conduct experiments to study the effectiveness of different recommendation algorithms in addressing cold-start issues on the two datasets. For evaluating the top-k items and friends recommendations, the target users who have less than 20 available items and social link information in total are selected. As there are not many interaction records between users and items available for cold-start cases, WRMF and RMFX model which are based on collaborative filtering, are not suitable for cold-start experiments. Thus, we compare our DGE model with other three recommender models that are able to leverage semantic effects and user relationships to recommend cold-start cases.



(a) Top-k items recommendation (b) Top-k friends recommendation

Figure 8: Recommendations for Cold-start Cases

The experimental results are shown in Fig. 8, from which we have the following observations: 1) our proposed DGE model still performs best consistently in recommending cold-start cases; 2) by comparing the recommendation results in Table 3 and 4, the *Recall* value of all recommendation algorithms decreases. For instance, the *Recall* value of DeepWalk rapidly drops from 35.1% to less than 4% for twitter item recommendation, while our model deteriorate slightly. This is because DeepWalk recommends items according to their content information such as hashtags and labels, while our method also considers the content similarities when training models and the potential relationships among all effects. This is to say, our model leverages not only user-item interactions, semantic effects and user relationships, but also the potential links between the features, when recommending cold-start items/users.

7. Conclusion

In this paper, a novel dynamic graph-based embedding model, DGE is proposed for real-time social recommendations. DGE jointly captures the temporal semantic effects, social relationships and user behavior sequential patterns in a unified way by embedding the heterogeneous user-item network into a shared low dimensional space for addressing the issues of temporal dynamics, cold start and context awareness in the social recommender system. To capture the semantic effect of different edge types, a transition probability matrix is devised and updated as the social network evolves. For efficiently handling large-scale social media streams, a parallel incremental learning algorithm and an efficient query processing technique are developed to generate top-k recommendations. Our recommendation process is based on the proximity of the related users and items while considering the freshness of the items. Evaluation on two different real-world datasets demonstrated the effectiveness of the proposed approach.

There are a series of future works we can do for the recommender system. First, we only consider the local context information (such as their neighbors) to learn vertex representations. There are still some global information can be employed, such as communities and groups. Second, more advanced deep learning models such as Convolutional Neural Networks can be explored for feature learning.

Acknowledgement

This work was supported by the Research Council of Norway (grant number 245469).

References

Agarwal, D., Chen, B.C., Elango, P., 2010. Fast online learning through offline initialization for time-sensitive recommendation, in: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 703–712.

Aggarwal, C.C., 2016. Recommender Systems: The Textbook. 1st ed., Springer Publishing Company, Incorporated.

Aggarwal, C.C., Wolf, J.L., Wu, K.L., Yu, P.S., 1999. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering, in: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 201–212.

Buccafurri, F., Lax, G., Nicolazzo, S., Nocera, A., 2016. A model to support design and development of multiple-social-network applications. Information Sciences 331, 99–119.

Chen, C., Yin, H., Yao, J., Cui, B., 2013. Terec: A temporal recommender system over tweet stream. Proceedings of the VLDB Endowment 6, 1254–1257.

Cremonesi, P., Koren, Y., Turrin, R., 2010. Performance of recommender algorithms on top-n recommendation tasks, in: Proceedings of the fourth ACM conference on Recommender systems, ACM. pp. 39–46.

De Meo, P., Nocera, A., Terracina, G., Ursino, D., 2011. Recommendation of similar users, resources and social networks in a social internetworking scenario. Information Sciences 181, 1285–1305.

Deshpande, M., Karypis, G., 2004. Item-based top-n recommendation algorithms. ACM Trans. Inf. Syst. 22, 143–177.

Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., Nejdl, W., 2012. Real-time top-n recommendation in social streams, in: Proceedings of the Sixth ACM Conference on Recommender Systems, ACM, New York, NY, USA. pp. 59–66.

Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M., Bhamidipati, N., 2015. Hierarchical neural language models for joint representation of streaming documents and their content, in: Proceedings of the 24th International Conference on World Wide Web, ACM. pp. 248–255.

Fagin, R., Lotem, A., Naor, M., 2003. Optimal aggregation algorithms for middleware. Journal of computer and system sciences 66, 614–656.

Hofmann, T., 2004. Latent semantic models for collaborative filtering. ACM Transactions on Information Systems (TOIS) 22, 89–115.

Hu, Y., Koren, Y., Volinsky, C., 2008. Collaborative filtering for implicit feedback datasets, in: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, Ieee. pp. 263–272.

Huang, Y., Cui, B., Zhang, W., Jiang, J., Xu, Y., 2015. Tencentrec: Real-time stream recommendation in practice, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM. pp. 227–238.

Jeh, G., Widom, J., 2003. Scaling personalized web search, in: Proceedings of the 12th international conference on World Wide Web, ACM. pp. 271–279.

Jia, Y., Song, X., Zhou, J., Liu, L., Nie, L., Rosenblum, D.S., 2016. Fusing social networks with deep learning for volunteerism tendency prediction., in: AAAI, pp. 165–171.

Koren, Y., 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 426–434.

Kouki, P., Fakhraei, S., Foulds, J., Eirinaki, M., Getoor, L., 2015. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems, in: Proceedings of the 9th ACM Conference on Recommender Systems, ACM, New York, NY, USA. pp. 99–106.

Le, Q.V., Mikolov, T., 2014. Distributed representations of sentences and documents., in: ICML, pp. 1188–1196.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013a. Efficient estimation of word representations in vector space. CoRR abs/1301.3781. URL: `http://arxiv.org/abs/1301.3781`.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013b. Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, pp. 3111–3119.

Mnih, A., Teh, Y.W., 2012. A fast and simple algorithm for training neural probabilistic language models. arXiv preprint arXiv:1206.6426 .

Morin, F., Bengio, Y., 2005. Hierarchical probabilistic neural network language model., in: Aistats, Citeseer. pp. 246–252.

Perozzi, B., Al-Rfou, R., Skiena, S., 2014. Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM. pp. 701–710.

Recht, B., Re, C., Wright, S., Niu, F., 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent, in: Advances in Neural Information Processing Systems, pp. 693–701.

Rendle, S., Schmidt-Thieme, L., 2010. Pairwise interaction tensor factorization for personalized tag recommendation, in: Proceedings of the Third ACM International Conference on Web Search and Data Mining, ACM, New York, NY, USA. pp. 81–90.

Shi, C., Zhang, Z., Luo, P., Yu, P.S., Yue, Y., Wu, B., 2015. Semantic path based personalized recommendation on weighted heterogeneous information networks, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, New York, NY, USA. pp. 453–462.

Stefanidis, K., Ntoutsi, E., Petropoulos, M., Nørvåg, K., Kriegel, H.P., 2013. A framework for modeling, computing and presenting time-aware recommendations, in: Transactions on Large-Scale Data-and Knowledge-Centered Systems X. Springer, pp. 146–172.

Subbian, K., Aggarwal, C., Hegde, K., 2016. Recommendations for streaming data, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM. pp. 2185–2190.

Sun, Y., Han, J., 2013. Meta-path-based search and mining in heterogeneous information networks. Tsinghua Science and Technology 18, 329–338.

Tang, J., Qu, M., Mei, Q., 2015a. Pte: Predictive text embedding through large-scale heterogeneous text networks, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM. pp. 1165–1174.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q., 2015b. Line: Large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, ACM. pp. 1067–1077.

Vinagre, J., Jorge, A.M., Gama, J., 2014. Fast incremental matrix factorization for recommendation with positive-only feedback, in: International Conference on User Modeling, Adaptation, and Personalization, Springer. pp. 459–470.

Wang, P., Guo, J., Lan, Y., Xu, J., Wan, S., Cheng, X., 2015. Learning hierarchical representation model for nextbasket recommendation, in: Proceedings of the 38th International ACM SIGIR conference on Research and Development in Information Retrieval, ACM. pp. 403–412.

Xie, M., Yin, H., Wang, H., Xu, F., Chen, W., Wang, S., 2016. Learning graph-based poi embedding for location-based recommendation, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM. pp. 15–24.

Yang, C., Liu, Z., Zhao, D., Sun, M., Chang, E.Y., 2015. Network representation learning with rich text information., in: IJCAI, pp. 2111–2117.

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, ACM, New York, NY, USA. pp. 283–292.

Zhang, J., Yu, P.S., 2016. Pct: partial co-alignment of social networks, in: Proceedings of the 25th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee. pp. 749–759.