Hussain Hussain

Graph Embeddings for Trust-based Recommender Systems

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Inf. Dr. Stefanie Lindstaedt

Graz, October 2019

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

One of the problems in user-based collaborative filtering recommender systems is the cold-start users. To overcome this problem, a viable solution is to collect as much information about the users of interest as possible. A particular example is a trust network that encodes explicit or implicit trust statements by a user indicating their trust of another user's taste or judgement. The aim of this work is to leverage the information obtained from trust networks associated with recommender systems by using graph nodes embedding. The work focuses on evaluating different graph embedding approaches on cold-start users recommendation in three benchmark datasets: Filmtrust, Ciao and Epinions. The embedding of each of these datasets is generated using several graph embedding approaches, then the recommendations to cold-start users are based on their obtained embedding. To this end, four different baselines and ten graph embedding approaches are considered. The graph embedding approaches are categorized into four families: (i) factorization-based approaches,(ii) random-walk-based approaches,(iii) deep-learning-based approaches and (iv) an edge-reconstruction-based approach. To analyze how certain methods outperform others, a statistical ranking test is performed on their results. Moreover, The effect of hyperparameters on the performance of certain methods is studied with further experiments and visualizations. Furthermore, two alternatives for dealing with directed graphs are analyzed: the bibliographic coupling graph and the cocitation graph. The obtained results show that the embedding approaches constantly outperform the baselines in terms of nDCG score and the statistical test suggests that random-walk-based embedding approaches achieve the highest ranks on average comparing to other families with node2vec achieving the highest rank overall. Finally, the tests on the special undirected versions of the graph show that the choice of ignoring the direction is a more feasible choice with higher accuracy and relatively less overhead.

Contents

List of Figures

List of Tables

List of Tables

Acknowledgements

I would like to sincerely thank my supervisor, Denis Helic, for his continuous support and his immediate help throughout this work. Besides, I would like to thank Tomislav Duricic for his help in the experiments and the evaluation, and Elisabeth Lex for her guidance on graph embedding approaches and on the directionality issue. I am also grateful to them for agreeing on incorporating a part of the experimental results from the other works we collaborated on in this thesis for further analysis and discussion.

This work is dedicated to my family who have always supported me.

Hussain Hussain
Graz, Austria, October 2019

1 Introduction

User-based collaborative filtering is one simple yet common paradigm in recommender systems. This technique depends on finding similar users based on users' ratings of items, that is, the user-item matrix. However, its dependence on the users' rating results in the problem of cold-start users, that are users who did not add enough ratings to build a specific profile or taste. In some systems, the user-item interaction is associated with the possibility of a user-user interaction through several activities, e.g., upvoting/downvoting a review, adding friends or issuing explicit trust statements of another user. This associating information provides more information about users and helps finding their preferences in order to generate meaningful recommendations.

In this work, we examine the example of the trust-based recommender systems. In such a system, a user can build a directed trust relationship with another user in case the former finds the latter's choices similar to what they might choose; which results in a trust network with directed edges. To exploit this network, similarities between users must be measurable, which can be accomplished by obtaining an embedding of the graph that preserves some network properties of interest. The obtained embedding is an encoding of each node (user) into a vector in a d-dimensional vector space, in which one can apply known vector similarity metrics, such as dot product, to decode similarities between pairs of users. Similarity between nodes in a trust network can be a strong indicator to similarity in preferences and hence a basis to form a collaborative filtering process afterwards.

The main goal of this work is to evaluate how different graph embedding approaches perform on the following task: *"Given a directed, possibly weighted trust network in a recommender system context, generate an embedding that can be used to compute similarities between users for a user-based collaborative filtering approach to generate recommendations to cold-start users."* The evaluated score is the accuracy in terms

of nDCG score.[1]

Another common issue in graph embedding context is the direction information of links in the network. We try to examine how to handle such type of graphs by evaluating on different versions of the given graphs as an undirected graphs.

The main research questions that this work tries to address can be formulated as follows

Q1) How do different families of graph embedding approaches perform on the trust-based cold-start recommendation problem?
Q2) How well can the directionality information, embedded within links directions, help obtaining a high-quality embedding?

To tackle the first question, three benchmark datasets from trust-based recommender systems are used for experiments: Filmtrust, CiaoDVD and Epinions, each of which contains a set of ratings as well as a set of directed trust statements building a trust network. Experiments are conducted using ten graph embedding approaches categorized into

1. Factorization-based approaches (LLE [47], LE [5], GF [2] and HOPE [43]).
2. Random-walk-based approaches (DeepWalk [45], node2vec [18] and role2vec [3]).
3. Deep-learning-based approaches (DNGR [9] and GraphSAGE [21]).
4. An edge-reconstruction-based approach (LINE [52]).

These are compared to the performance of four simple baselines. The resulting embedding of each of the studied methods is used to find the k-nearest neighbors (k-NN) of each user and then generate the recommendations for a certain user based on the ratings of their k-NN.

To address the second question, the same datasets are used to obtain the bibliographic coupling graph and the cocitation graph, and then generate the embedding of a subset of the aforementioned embedding approaches, and analyze how the accuracy changes accordingly.

The results of performed experiments mainly suggest that

[1] The terms 'accuracy' and 'nDCG' may be used interchangeably in this work considering that they refer to the same concept in this very work.

1) Graph embedding approaches constantly achieve higher accuracy on the considered task with the random-walk-based approaches in particular outperforming the other embedding approaches on average.
2) Ignoring the links direction is a feasible simple solution that results in a good performance in the environment of trust network and in the course of the studied task while incorporating the link direction results in an overhead with slight or no increase in recommendation accuracy.

The next chapter (Chapter 2) gives a background of the problem, introduces a terminology and reviews work conducted on network-based recommender systems as well as graph embedding. Chapter 3 explains the two-phase pipeline this work follows and discusses graph embedding approaches which are considered for experimentation. Chapter 4 analyzes the datasets, shows the hyperparameter settings and explains the conducted experiments and the statistical ranking test as well as the dimensionality and the directionality experiments. Chapter 5 summarizes and visualizes the results of the conducted experiments. Chapter 6 highlights the interesting results and the main findings of this work, and analyzes some observations through further experimentation. Finally, chapter 7 concludes the thesis and proposes directions for future work.

A part of this thesis will be submitted to ECIR 2020 conference as the thesis is a part of a larger body of joint work with other researchers. The majority of the conducted experiments here are used for evaluation and discussion in that work.

2 Related Work

2.1 Background

2.1.1 Graph theory basics and terminology

A graph (or a network) $G = (V, E)$ can be defined by two sets:

- a set of nodes (vertices) V[1], which represent entities in our data, and
- a set of edges (links) $E \subseteq V^2$ which carry the relationship information between pairs of nodes.

Edges can carry relationship (e.g., knowledge graphs), but in this work this relationship is only considered to be a scalar.

A graph can be either

- unweighted, where an edge can have a weight in $\{0, 1\}$, or
- weighted, where the edge can have any weight in $\mathbb{R}$.

A graph can be either

- directed, where the edge (u, v) is a sorted set, and it does not provide any knowledge about (v, u) or even whether such an edge exists, or
- undirected, where the edge (u, v) refers to the same entity as the edge (v, u).

The graph $G = (V, E)$ is usually represented as a matrix A of size $n \times n$ called the adjacency matrix. For a weighted graph, we have $A \in \mathbb{R}^{n \times n}$, and for an unweighted graph, we have $A \in \{0, 1\}^{n \times n}$. For an undirected graph, we have $A = A^T$, whereas this statement does not hold for a directed graph. Figure 2.1 shows an example of a directed unweighted graph with its adjacency matrix.

[1] Conventionally, $|V| = n \in \mathbb{N}$.

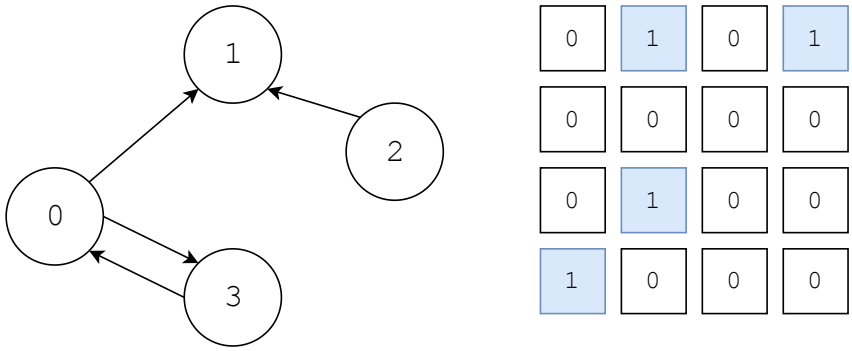


Figure 2.1: An example graph with its corresponding adjacency matrix.

Note: *the terms 'graph' and 'network' are both used in this work to refer to the same entity $G = (V, E)$. The term 'graph', however, is mostly used in order not to confuse the term 'network' with the short name of 'artificial neural networks' which is used often in this work.*

Random walks

In its simple form, a random walk W starting at a node u_0 is a sequence of nodes $W = (u_0, u_1, ..., u_{|W|-1})$, where each node $u_i : 0 < i < |W|$ is sampled from a uniform distribution over the set of the node u_{i-1} neighbors (that is $\mathcal{N}(u_{i-1})$). This reduces to a one-step transition of a Markov chain [55, 32].

Random surfers transform the graph into a set of walks, each of which is a list of nodes generated from a single random walk, stating at a certain node. The walks can be used to get co-occurrences between nodes, which can be used as a similarity measure, but the co-occurrence itself depends on the parameters and the bias of the random walk as some high-order random walk methods work.

From directed to undirected

As some algorithms on graphs work better on or only exist for undirected graphs [24], it seems viable to turn the directed graph into an undirected one. One of the possibilities is to ignore the link directions, which is simple to implement and keeps the adjacency matrix sparse, but it causes a loss of direction information. Another

solution is to use the *cocitation or bibliographic coupling graphs*. The cocitation matrix is computed by $C = A^T A$ where the element $C_{u,v}$ represents the number of nodes that have outgoing edges to both u and v. The bibliographic coupling matrix is computed by $B = AA^T$ where the element $B_{u,v}$ represents the number of nodes that have incoming edges from both u and v. These alternatives make the matrix denser but they preserve direction information implicitly.

2.1.2 Graph node embedding problem

Given a graph $G = (V, E)$, a graph node embedding is a mapping ENC : $V \rightarrow \mathbb{R}^d$, where $d \ll n$ and ENC is usually referred to as *the encoder* function [22]. The goal is to obtain such a mapping which preserves certain properties of the graph in order to make it more efficient to discover information about the relationships between nodes and infer knowledge about their hidden features using known mathematical similarity metrics in a Euclidean space, such as dot product and cosine similarity. For a node $u \in V$, its embedding is denoted as $z_u = \text{ENC}(u) \in \mathbb{R}^d$

Usually, a set of parameters Θ is associated with the mapping ENC which makes the function trainable. A simple example of the encoder function is the shallow embedding encoder, which most transdutive graph node embedding approaches use. It is calculated by a shallow lookup:

$$z_u = \text{ENC}_\Theta(u) = \Theta \cdot I_u, \tag{2.1}$$

where $\Theta \in \mathbb{R}^{d \times n}$ is the embedding (encoding) parameter, $I \in \mathbb{R}^{n \times n}$ is the identity matrix of size n, and I_u is just the $u-$th column in I. This means that a node embedding is just a column in Θ and the training directly optimizes the embeddings themselves, which makes these methods inherently transductive [22]. This idea is visualized in figure 2.2.

Note: *The term 'graph node embedding' is sometimes referred to simply as 'graph embedding'. In this work, both terms are used interchangeably, with no reference to other types of graph embeddings such as whole-graph embedding or subgraph embedding unless stated explicitly.*

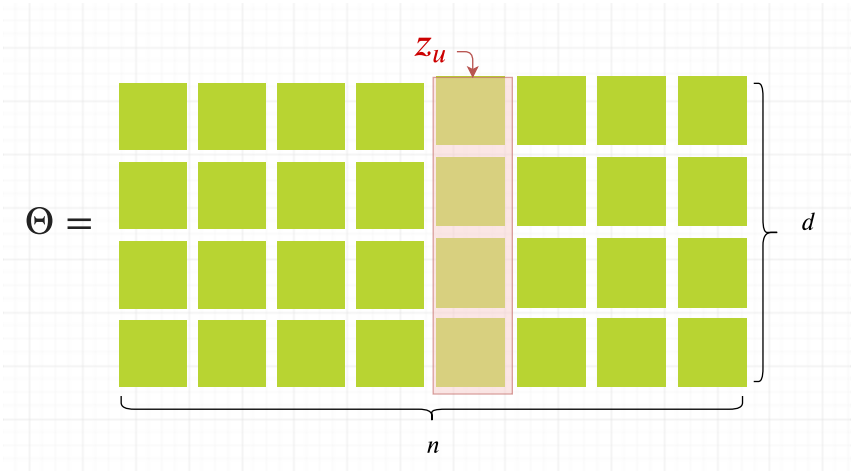


Figure 2.2: Shallow lookup embedding.

2.1.3 Trust-based recommender systems

In trust-based recommender systems, we are given a user-item rating matrix, R along with a **directed**, **probably weighted** trust network $G = (V, E)$, represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$. An element $A_{u,v} : (u, v) \in V^2$, represents whether user u trusts user v. Different platforms define this trust relationship in different ways: some explicitly, some implicitly. Mainly, one can intuitively assume that a user u will only trust user v if they think v's choices of items are potential choices for them.

The goal is to generate recommendations, i.e., filling up the blanks in R, using information from both matrices.

User-based collaborative filtering. User-based nearest neighbor recommendation [25] is a form of collaborative filtering which aims to provide recommendations to an active user u by finding the most similar users to u. This similarity can be depending on their ratings of items or any other relevant information such as their position/role in the trust network in the trust-based case. One of the main problems with the classic approach which depends on the user's ratings is what is known as the **cold-start problem**.

cold-start users. The user cold-start problem is a special case of the sparsity problem in recommender systems, and it refers to cases where user u have not rated many (or any) items yet. This makes the job of generating recommendation for this user more difficult as no much inference can be done about the user's preferences. Additional data about the users, such as the associated social/trust network, can ease the problem, and this is exactly the case this work deals with.

2.2 Literature review

This work aims to evaluate different graph embedding techniques on the task of recommending items in a trust-based recommender systems. Following is a review of literature related to the network-based recommender systems and graph node embedding.

2.2.1 Network-based recommender systems

An earlier work on trust-based recommender systems by Massa and Avesani [35] uses PageRank global metric of the trust network to build an estimated trust matrix, that represents similarity between users in the network, which is then used along with a similarity matrix from the ratings information to predict unknown ratings. Ma et al. [33] use matrix factorization for social recommendation which utilizes both the adjacency matrix and the user-item matrix simultaneously. Andersen et al. [4] propose multiple approaches to the problem: a random-walk based recommender, a majority-of-majority system that only applies to directed acyclic graphs, a minimum-cut system and a personalized PageRank system.

Ma et al. [34] later interpret the difference between social-based and trust-aware recommender systems. They use the social network as a regularizer that refines the matrix factorization based target recommender system.

A more recent work by Li et al. [30] uses the overlapping community information as a regularization term for user-item matrix factorization. Seo et al. [50] calculate an expected friendship strength between users and use the explicit social network to generate personalized recommendations. Finally, Duricic et al. [11, 12] use Katz proximity measure in the trust network to generate a similarity matrix and then

produce recommendations with collaborative filtering aiming to overcome the cold-start problem in recommender systems.

2.2.2 Graph embedding

Graph embedding problem has drawn attention in the past two decades, mostly in the 2010s. Earlier approaches, such as LLE [47] and LE [5], depended on factorization of the adjacency matrix but are not scalable as they need $O(|E|d^2)$ time. Later, Graph Factorization [2] was proposed as a more scalable factorization-based approach with a complexity of $O(|E|d)$. The approaches mentioned so far aim to preserve fist-order proximity, by embedding nodes close to each other in the target vector space in case they share edges in the graph. GraRep [8] extendes the preserved property to a defined $k^{th}-$order proximity but had a complexity of $O(|V|^3)$. HOPE [43] also preserves high-order proximity but generalized the similarity matrix to include known measurements such as Katz similarity [27], personalized PageRank [51] and Adamic-Adar [1]. LINE [52] only aims to preserve $1^{st}-$ and $2^{nd}-$order proximity. LINE uses a probabilistic based loss function and is not strictly classified as a matrix factorization approach.

Another line of graph embedding approaches, random walk based ones, was spawned by DeepWalk [45] which generates a set of random walks and applies Skipgram model [38] to generate embeddings for nodes similar to generating embeddings for words in word2vec using Skipgram model. Node2vec [18] follows the same idea of DeepWalk but uses a 2^{nd}-order random walk to control how deep and how broad the random surfer is moving. A more complicated framework, Struc2vec [46], focuses on the structural similarity regardless of node attributes or position to build a multi-layer graph, from which a contexts of nodes are built using random walks, then a latent representation is learned for each node. A more recent approach, role2vec [3], introduces role-based random walks that traverse a mapping of the nodes to a their structural role.

The rapid advancement deep learning in the recent years has also made its way to graph representation learning. Earlier neural-networks-based approaches [16, 37, 48] used recurrent neural networks to learn a node's representation using message passing between nodes through the RNN. Later, this was extended by Li et al. [31] to use Gated Recurrent Units.

Later on, several works attempted to adapt convolutional neural networks to graph data. Deep convolutional neural networks are mainly applied to grids with predefined neighborhood, e.g. 4-connectivity in images, which makes the convolution operator easily defined as opposed to graphs where the neighborhood is irregular and differs from a node to another which is a challenge for deep convolution-based approaches. Convolution-based models are, however, proven to work well so far. Some work on these models aimed to obtain the convolution through linear operators in the spectral domain. For example, Bruna et al. [6] utilize the graph Laplacian as spectral representation of the graph and aim to find the spectrum of the weights. Similarly, the GCN model [28] use multiplication in the spectral domain and produce a simple, multi-layered propagation rule. More recently, spatial convolutional approaches were proposed to overcome the complexity of the spectral approaches and also to introduce an inductive setting of representation learning on graphs, in contrast to most previously mentioned methods. For example, with GraphSAGE [21], neighborhood information is propagated through fixed-size sampling and symmetric aggregation. In contrast, GAT [53] introduces an attention-based architecture that allows asymmetric aggregation of the neighborhood and access to the entire neighborhood instead of a fixed-size sample. LCGL [14] aims to enable regular convolutional operations on graph data. They propose $k-$largest node selection that transforms each node's aggregated neighborhood to a 1-D grid on which a 1-D CNN is applied. PATCHY-SAN [42] had a different approach of applying CNNs on graphs: it define a sequence of nodes for which a receptive field is created and determines an embedding that preserves structural roles.

Other deep learning based approaches utilize the deep autoencoders which were adapted to work on graphs by using an input feature vector representing the similarity between the node and other nodes in the graph. DNGR [9], for example, applies a stacked denoising autoencoder on normalized (with PPMI) nodes' co-occurrence vectors, which are obtained through random walks. SDNE [54], define a similar unsupervised component with the adjacency matrix in addition to another supervised component that exploits the first-order proximity of some pairs of nodes, that is used to refine the learned representations. These approaches, however, are expensive since they take a global neighborhood of the node as an input.

Several recent works have surveyed the existing graph embedding approaches. Some of them went a step further to test these methods on different down-stream machine learning tasks and visualize the embeddings in 2 dimensions to extract some insights.

Hamilton et al. [22] decompose the graph embedding problem into 4 main parts as follows

(i) an encoder that transforms the network representation into the latent vector representation,
(ii) the graph similarity measure which defines how two nodes $(u, v) \in V \times V$ are similar based on the network information, e.g. the adjacency/weight matrix, edge information
(iii) the decoder which can be seen as a latent vector (dis)similarity measure, e.g. dot product, cosine similarity, euclidean distance
(iv) the loss function which defines the penalty on the difference between the graph similarity and the latent vector similarity, and can be simply an absolute difference, a squared difference, etc.

This scheme makes it simpler to try different combinations of components and interpret the correlation between the results of different known approaches. They classify the embedding approaches into deep and shallow with the shallow approaches always having the encoder function as a look-up table. Hamilton et. al [22] do not only study methods of graph nodes embeddings but also whole graph embedding and subgraphs embedding which are not considered in this work.

Cai et al. [7] provide a taxonomy for the graph embedding by problem settings, i.e. input graph and output embedding, as well as by techniques which we discuss later on. They categorize the input graph types into homogeneous graphs where each edge/node has a single type, heterogeneous graphs where multiple types per node/edge are possible, graphs with auxiliary information (such as node labels/features, edge attributes or knowledge bases) and graphs constructed from non-relational data, e.g. constructing edges between documents based on a calculated similarity measure by applying KNN. The output embedding is either node, edge, hybrid (a combination such as node+edge embedding or subgraph embedding [57]) or whole-graph embedding. The embedding techniques are split into matrix factorization, deep learning, edge reconstruction, graph kernel and generative models.

Zhang et al. [59] study the network representation learning methods and provide a hierarchical taxonomy of methods based on supervision (unsupervised/semi-supervised), preserved properties (structure preserving/content augmented) and information sources (network structure/vertex labels and attributes). They show the main advantages/disadvantages of each family of approaches and focus on

how local/global the considered structural information are and further perform a complexity analysis. They categorize 22 different benchmark datasets and compare seven unsupervised algorithms on vertex classification and vertex clustering with 7 datasets.

Goyal and Ferrera [17] categorize the embedding methods into (i) matrix factorization based, (ii) random walk based and (iii) deep learning based. Then they study the performance of more than 10 methods on four different tasks, i.e. node classification, graph reconstruction, clustering, and visualization, on corresponding datasets. They further study the effect of dimensionality on the performance.

Finally, Wu et al. [56] study graph neural networks in general and explain the overlap between this term and the graph embedding term. They define a taxonomy for graph neural networks by categorizing them into graph convolutional networks, graph attention networks, graph auto-encoders, graph generative networks and graph spatial-temporal networks. The last two are not assumed to work as graph embedding methods.

3 Approach

In this chapter, we define the task of interest and the recommendation approach which is composed of two steps (obtaining the embedding and generating the recommendation). Then we discuss the used graph embedding approaches and their most interesting features.

3.1 Task definition and strategy

The goal of our experiments is to evaluate different graph embedding approaches on the cold-start problem in recommender systems. We assume having a set of users and a set of items, where users can rate the items, as well as trust relationships between users.

3.1.1 Cold-start users

A cold-start user is a user who did not rate many items in a sense that makes it hard for the system to infer information about the user in order to produce recommendation for them. In this work, we select a threshold of 10 to define the cold-start users; in other words, we consider users that have 10 or less ratings for items to be **cold-start users**.

In addition to rating items, a user u can issue a trust statement of another user v, in case u finds that v's ratings are consistently valuable, which forms a network of directed trust statements. This trust relationship can be explicitly set by u or implicitly inferred based on other information in the system. For example, if users had the ability to provide written reviews of items, and other users could rate these reviews, we can infer a trust (or a distrust) relationship from a user u to a user v

based on the the ratings u gives to the written reviews of v. Therefore, the trust statement in such a system intuitively implies a similarity in interests or preferences of the two users.

3.1.2 Problem statement

To tackle the cold start problem, we aim to utilize this network using graph node embedding approaches to find a low-dimensional vector representation of users and then apply collaborative filtering to generate recommendations for cold-start users. We are given

1. a set of ratings as a relationship between items and users and
2. a directed graph $G = (V, E)$ where nodes $u \in V$ represent users and edges represent trust statements between users, i.e., an edge $(u, v) \in E$ means user u trusts user v.

3.1.3 Recommendation strategy

Our methodology consists of two main phases:

1. Generating an embedded representation for each node (user) $u \in V$ with a size of d dimensions.
2. Applying collaborative filtering to find similar users in the resulting d-dimensional space and generate recommendations to target users on this basis.

For the first phase, we only utilize the network and don't use the set of ratings. Our considered embedding approaches are therefore merely structural with no nodes features and unsupervised with no feedback about the embdding quality. As we are only interested in the embedding of unsigned networks that only have non-negative edges, and since most datasets contain no explicit distrust statements publicly for privacy reasons, we will only consider positive trust statements and hence unsigned networks.

For the second phase, we use a user-based collaborative filtering technique with a k-nearest neighbors algorithm. The similarity between two users is calculated using the dot product of their representations (embeddings) in the d-dimensional vector

space, generated by the first phase, which produces a similarity matrix between users.

Through our experiments, we consider two types of users:

- Cold-start users who have rated $1 \leq q \leq 10$ items. These users are considered as a target test set for our evaluations.
- Warm users who have rated $q > 10$ items. These users are considered as a target validation set to perform our hyper-parameter optimization.

There is no need for a training set as the embedding approaches are completely unsupervised.

For a target user u with q rated items, the 80 most similar non-target users are found from the similarity matrix, and then used to generate a ranked list of items (top-q items) to recommend to u. The ranks are based on a score which is a linear combination of the neighbors' ratings times their similarities to u, computed as follows for an item i:

$$\text{score}(u, i) = \sum_{v \in \text{k-NN}(u)} \text{sim}(u, v) \cdot \mathbf{1}(v, i), \tag{3.1}$$

where the $\mathbf{1}(v, i)$ is and indicator that equals 1 if user v provided a rating for item i and is 0 in case user v has no rating for i. These items are then sorted based on these scores, and the top-q items are recommended to u.

When generating recommendations for a warm user u in the validation phase, we randomly remove 10 of u's ratings, recommend the top-10 ranked items (which are not contained in the remaining items rated by u) to u according to Equation 3.1. In this case, k-NN(u) contains other warm users whose ratings are completely visible while focusing on u. On the other hand, when generating recommendations for a cold-start user u with $q \leq 10$ ratings, we remove all of their ratings and recommend the top-q ranked items to u according to Equation 3.1. In the cold-start case, all the ratings by all cold-start users are not visible and the k-NN(u) set contains only warm users.

3.2 Graph embedding approaches

The graph node embedding approaches, considered for experiments, are explained in this section to the level that serves the purpose of the experiments. This work follows the categorization of Goyal and Ferrera [17] which goes as follows.

3.2.1 Factorization-based approaches

Factorization-based methods aim to factorize a matrix representing a similarity measure between pairs of nodes. The similarity measure can be the, possibly weighted, adjacency matrix A (with elements $a_{i,j}$ itself or another similarity matrix, e.g. Katz similarity, PageRank, etc.). Factorization-based methods were generally proposed earlier than other approaches.

Locally Linear Embeddings. LLE [47] treats each node representation as a linear combination of its neighbors $z_u \approx \sum_{v \in (N)(u)} a_{u,v} z_v$ and aims to minimize the difference between the representation and this linear combination as follows

$$\sum_{u \in V} |z_u - \sum_{v \in (N)(u)} a_{u,v} z_v|^2,$$

where z_u is the embedding of node u.

Laplacian Eigenmaps. LE [5] aims to minimize the sum of squared differences between the representations of pairs of nodes, with the sum being weighted by the edge weight $a_{u,v}$ in the adjacency matrix, which is 0 in case no edge exists. In particular, the algorithm minimizes the cost function

$$\sum_{(u,v) \in V^2} |z_u - z_v|^2 a_{u,v}.$$

Note that this method uses the L-2 norm ($|.|^2$) as a decoder, the 1^{st}-order proximity as a similarity in graph and the multiplication as a loss function for a pair of nodes.

By removing degenerate solutions and translational invariance, both LLE and LE reduce to an eigenvalue decomposition problem.

Graph Factorization. GF [2] follows a similar approach to LLE[47] and LE [5] but improves the efficiency by minimizing over existing edges. It uses the dot product as a decoder and the L-2 norm as a loss function per edge. Additionally, it introduces a regularization term that penalizes representations with big L-2 norms. The objective function turns out as

$$\frac{1}{2}\sum_{(u,v)\in E}|z_u^T z_v - a_{u,v}|^2 + \frac{\lambda}{2}\sum_{u\in V}|z_u|^2 \tag{3.2}$$

which is minimized using stochastic gradient descent that iterates until the vector representation converges. This algorithm runs in $O(|E|d)$ time comparing to $O(|E|d^2)$ for both LLE and LE.

HOPE. HOPE [43] defines a more general cost function by utilizing a similarity measure S that can be defined by the user as long as it can be represented as $S = M_g^{-1} \cdot M_l$. The authors investigate different similarity measures, namely, Katz similarity [27], Rooted PageRank [44], Common neighbors and Adamic-Adar (AA) [1] and fit them into that formulation and then solve the minimization problem

$$\min ||S - Z^s \cdot {Z^t}^T||_F^2, \tag{3.3}$$

where $Z^s, Z^t \in \mathbb{R}^{|V|\times d}$ are, respectively, the source and the target embedding vectors both with representation size d.

3.2.2 Random-walk-based approaches

Several approaches utilize the lists resulting from random walks as they can be regarded as a regular (linear) form of data that represent context or similarity and can be fed to machine learning models.

DeepWalk. DeepWalk [45] shows that techniques used to model natural language can be also used to model community structure. It uses a stream of short random walks in order to learn a latent representation $\Phi(v)$ for each node $v \in V$. For a node v_i in a short random walk $v_1, v_2, ..., v_{len}$, the language model, namely

the skip-gram model [38], maximizes the probability of v_i's context given the latent representation of v_i which is $\Phi(v_i)$:

$$\max_{\Phi(v_i)} \Pr(\{v_{i-k}, v_{i-k+1}, ..., v_{i+k-1}, v_{i+k}\} | \Phi(v_i))$$

where the unordered set $\{v_{i-k}, v_{i-k+1}, ..., v_{i+k-1}, v_{i+k}\}$ is the context of node v_i in the corresponding random walk, and $2k + 1$ is the window size.

Skip-gram algorithm is used to update the representation after obtaining each random walk, and hierarchical softmax [39, 40] is used to approximate the probability distribution.

node2vec. Grover et al. [18] follow the same scheme of DeepWalk [45] but extend the definition of the random walk to a 2-nd order random walk that has extra memory which is the predecessor node. Consider the current node $u_i \in V$, the predecessor would be $u_{i-1} \in V$ then the (unnormalized) probability of picking a node $v \in V$ would be

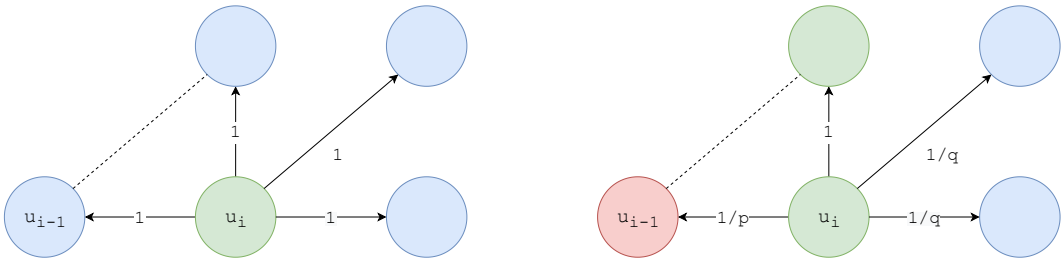


Figure 3.1: A comparison between (left) 1^{st} order random walks as in DeepWalk and (right) 2^{nd} order random walks as in node2vec. The labels on the edges get multiplied with the corresponding weights and then normalized to be a probability distribution.

$$\alpha_{u_{i-1},u_i}(v) = \begin{cases} 1/p & \text{if } d(u_{i-1}, v) = 0 \\ 1 & \text{if } d(u_{i-1}, v) = 1 \\ 1/q & \text{if } d(u_{i-1}, v) = 2 \end{cases} \tag{3.4}$$

where $d(u, v)$ is the shortest distance between nodes u and v and p, q are parameters of the random walk which define how biased it would be towards depth-first or

breadth-first search. Mainly, relatively higher p would mean it is less likely for the random surfer to go back to a node that it just visited, which pushes the walks towards depth. Conversely, relatively higher q means that the surfer is less likely to explore outside and favors nodes that are closer to the predecessor node. node2vec uses negative sampling to approximate the probability distribution.

Role2vec. Role2vec [3] introduces the concept of random walks which are tied to a function that maps each vertex to a structural role instead of the vertex identity, then embeddings are learned for types and not nodes. This model can be applied to new nodes, and can take vertex features as input.

The mapping function is either defined manually by the user or learned automatically by means of low rank factorization of the feature matrix and then applying k-means to map them to clusters (types). The attributed random walks are then generated simply by mapping each vertex in a random walk, which follows the description of random walks in node2vec, to its type.

Role2vec, node2vec and DeepWalk use an asymmetric dot-product-based decoder

$$\text{DEC}(u, v) = \frac{e^{z_u^T z_v}}{\sum_{k \in V} e^{z_u^T z_k}} \approx p(v|u), \tag{3.5}$$

where $p(v|u)$ models the probability of visiting v in a random walk starting at u.

They use a cross-entropy loss function

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \log(\text{DEC}(u, v)), \tag{3.6}$$

where $\mathcal{D}$ is sampled from the distribution $p(.|.)$. This loss is approximated through hierarchical softmax in DeepWalk and through negative sampling in role2vec and node2vec [22].

3.2.3 Deep-learning-based approaches

We consider one autoencoder-based method, and inductive convolutional-based methods that are included as different aggregators in GraphSAGE.

DNGR. DNGR [9] starts with random surfing to build a probabilistic co-occurrence matrix of nodes. Then it calculates the PPMI matrix of it. A stacked denoising autoencoder is then used to embed the nodes, which results in a non-linear mapping to the low-dimensional vector representation unlike the SVD-based solutions that yield linear mappings. The PPMI matrix proposed by [29] is a version of PMI (pointwise mutual information) matrix [10] which is defined as

$$\mathrm{PMI}_{u,v} = \log(\frac{\#(u,v)|D|}{\#(u)\#(v)}) \tag{3.7}$$

where $\#(u)$ is the count of occurrences of u, $\#(u,v)$ is the count of co-occurrences of u, v and $|D| = \sum_u \sum_v \#(u,v)$. The PPMI (*positive shifted* PMI) is simply $\mathrm{PPMI}_{u,v} = \max(0, \mathrm{PMI}_{u,v})$.

GCN. Kipf et al. [28] suggest Graph Convolutional Networks (GCN), a multi-layered model that can be defined with the recursive propagation rule

$$Z^{(i+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}Z^{(i)}W^{(i)}) : 0 < i \leq h \tag{3.8}$$

where $Z^{(i)} \in \mathbb{R}^{N \times F_i}$ is the output of the i-th layer, $\tilde{A} = A + I_N$ (the adjacency matrix with added self-loops), $\tilde{D}_{u,u} = \sum_v A_{u,v}$, σ is a non-linearity unit, $W^{(i+1)} \in \mathbb{R}^{F_i \times F_{i+1}}$ is a trainable weight matrix for layer i, h is the number of the hidden layers and $Z^{(0)}$ is the input feature matrix. The embedding of the nodes is represented by $Z^{(h)} \in \mathbb{R}^{N \times F_h}$. The input feature matrix in our case is just the identity as we do not consider node features.

The above formula is (Eq. 3.8) can be realized by a 1-localized approximation of Chebyshev polynomials as suggested by Hammond et. al [23] and then approximating this by sharing weights between the 0- and 1-order of the polynomial in order to accelerate the training process. By stacking multiple layers, the signal propagates in the way the network learns.

Since the multiplication in the update rule happens in the spectral domain, it is equivalent to a convolution in the spatial domain. Hence, the learned weights are actually the spectral representation of the filters.

GCN was not strictly used in this work with this spectral representation, but rather with the spacial equivalent that is realized in GraphSAGE.

GraphSAGE. Proposed by [21], GraphSAGE (Graph SAmple and aggreGatE) model consists of several layers, each of which performs, as the name suggests, sampling and aggregating on the incoming information from the previous layer to generate a vector representation for each node.

The sampling operation is uniform on the neighborhood of a node, but the number of samples can differ from a layer to another. The suggested aggregators are (i) mean-based aggregators (ii) LSTMs and (iii) pooling aggregators. A modified version of the mean-based aggreagator reduces to an inductive variant of GCN by [28]. The aggregated vectors are multiplied with a weight matrix and then a non-linearity is applied to that result. This is represented by the following propagation rule for each layer $0 < i \leq L$:

$$\begin{aligned} \hat{h}_u^{(i+1)} &= \text{AGG}_{v \in \mathcal{N}(u)}^{(i+1)}(h_v^{(i)}) \\ h_u^{(i+1)} &= \sigma(W^{(i+1)} \cdot \text{CONCAT}(h_u^{(i)}, \hat{h}_u^{(i+1)})), \end{aligned} \tag{3.9}$$

where AGG is the aggregation function, CONCAT denotes vector concatenation, $\mathcal{N}(u)$ is node u's sampled neighborhood, $\hat{h}_u^{(i+1)}$ is an intermediate representation of the aggregation of the node u's neighborhood, $h_u^{(i)}$ is the representation of node u in layer i, L is the total number of layers, σ is a non-linearity unit and $W^{(i)}$ is the learnable shared weight matrix in layer i.

The weight matrix, as well as the LSTM and the pooling aggregators, is trainable. All aggregators are meant to be symmetric with respect to the input samples, which is why the LSTMs are fed with a random permutation due to their sequential nature.

The loss function for the unsupervised case promotes the similarity between two co-occurring nodes and penalizes the similarity between a node and several negative samples. To use this loss function, a fixed-length random walk is applied to get a co-occurrence relationships between pairs of nodes. This yields the following loss function

$$J(z_u) = -\log(\sigma(z_u^T z_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-z_u^T z_{v_n})), \tag{3.10}$$

where v is a vertex co-occurring with u in a random walk, σ is sigmoid function, P_n is a negative sampling distribution and Q is the number of negative samples.

3.2.4 Edge reconstruction based approach

LINE. LINE [52] model aims to preserve first-order and/or second-order proximity between pairs. It is classified by [7] as an edge-reconstruction-based approach and it is always compared to node2vec [18] and DeepWalk [45].

The empirical first-order proximity can be defined on an undirected (weighted) graph $G = (V, E)$ with weight matrix $A \in \mathbb{R}+^{|V|\times|V|}$ between two nodes $(u, v) \in V \times V$, as $\hat{p}_1(u, v) = \frac{a_{u,v}}{\sum_{(i,j)\in E} a_{i,j}}$, which is 0 if there is no edge. To deal with directed graph, we just ignore the direction of links here to get a symmetric A. The first-order proximity between their vector representations z_u, z_v is

$$p_1(u, v) = \frac{1}{1 + \exp(-z_u^T.z_v)} \tag{3.11}$$

which, as for the empirical one, defines a probability distribution $p_1(., .)$ over the space $V \times V$.

The empirical second-order proximity, however, can be defined on both directed and undirected graphs as $\hat{p}_2(v|u) = \frac{a_{u,v}}{\sum_{(u,i)\in E} a_{u,i}}$. This value should be close to the second-order proximity of their vector representations which consists of two roles for each node: z_u is the representation of u as a vertex and z'_u is its representation as a context.

The proximity is then defined as

$$p_2(v|u) = \frac{\exp(< z'_v, z_u >)}{\sum_{k\in V} \exp(< z'_k, z_u >)} \tag{3.12}$$

which again, as for the empirical second-order proximity, defines a conditional probability distribution $p_2(.|u)$ over V.

Both types of proximity (first- and second-order) are preserved by minimizing the KL-divergence between the empirical and the resulting distributions. This gives the following loss function for the first-order proximity case:

$$\mathcal{L}_1 = -\sum_{(u,v)\in E} a_{u,v} \log p_1(u, v) \tag{3.13}$$

and the following one for the second-order proximity case:

$$\mathcal{L}_2 = \sum_{(u,v)\in E} a_{u,v} \log p_2(v|u). \tag{3.14}$$

The relationship between LINE and random-walk-based approaches is the probabilistic decoder and loss function. The major difference is that LINE does not generate node context as a random walk, but uses the explicit proximity instead.

4 Experiments

This chapter explains the experimental setup by introducing the datasets considered for the cold-start problem, listing the considered baselines and describing the combinations of hyperparameters tested for each of the studied embedding approaches, explains the evaluation protocol and introduces further experiments that are conducted on a narrower scale.

4.1 Datasets description

The experiments are conducted on three datasets: Filmtrust, CiaoDVD and Epinions, each of which has a set of ratings done by users to evaluate items, and a trust network that connects users. Some facts and statistics about the used datasets are presented bellow.

Filmtrust. Filmtrust [15] was a website that provides movie recommendations to users. It allows them to rate the movies and provide a trust rating for other users. This website was created to exploit the trust relationships between users in the context of recommendation. Users can give ratings to movies, which vary between *half a star* and *4 stars*. They can follow other users and provide trust ratings for them. This rating is meant to represent how likely the trustor would watch a movie that was highly rated by the trustee. The rating can be accompanied by a free-text review, which is not considered in this work.

The considered dataset [20], that was crawled from the entire website in 2011, consists of $1,508$ users and $2,071$ movies with $35,497$ ratings. In addition, a trust network that have 874 of total users is provided. This network has $1,853$ directed edges representing trust statements (see Figure 4.4c for degree distribution).

CiaoDVD. Ciao[1] is an online-shopping website that allows users to write reviews about items or rate them in addition to rating other people's reviews. It allows the user to create a 'Circle of Trust' which consists of the users whose reviews are consistently helpful for that specific user. The ratings of these reviews can vary from 0 to 5. These trust relationships again form a network of trust.

The CiaoDVD dataset [19] is crawled in December 2013 from the DVD category in the website. It contains $17,615$ users that have rated $16,121$ items with a total of $72,665$ ratings. The ratings file also includes the rating date and the movie genre; both will not be visible to the training or the evaluation phase. A trust network is built depending on the circle of trust of the included users. However, in this dataset, only $4,658$ of $17,615$ users have trust connections in the network, and the rest are isolated. The network has $40,133$ directed edges. Figure 4.2 shows the degree distribution of CiaoDVD network. Lastly, a set of review ratings are available in the dataset too, but this will not be visible to our approach either.

Epinions. Epinions[2] is another online-shopping website, which has the same ideas of the reviews rating and the 'Circle of Trust' described for Ciao.

The considered dataset [36] contains $49,288$ users with $139,738$ items and $664,824$ user-item ratings. The network contains all $49,290$ users with $487,183$ directed edges. The degree distribution of Epinions is shown in Figure 4.3. Table 4.1 summarizes the statistics of the datasets.

4.1.1 Dataset split

We split each of the three datasets as pointed out in Chapter 3 into two subsets: cold-start users and warm users, with warm users representing a validation set for our hyperparameter settings search, and cold-start users representing the test set of our experiments, on which we report the accuracy for the selected models. Table 4.2 shows the number of cold-start and warm users in each dataset and highlights their numbers within the trust network. Users who have not provided any ratings are neither cold-start nor warm users.

[1] www.ciao.co.uk
[2] www.epinions.com

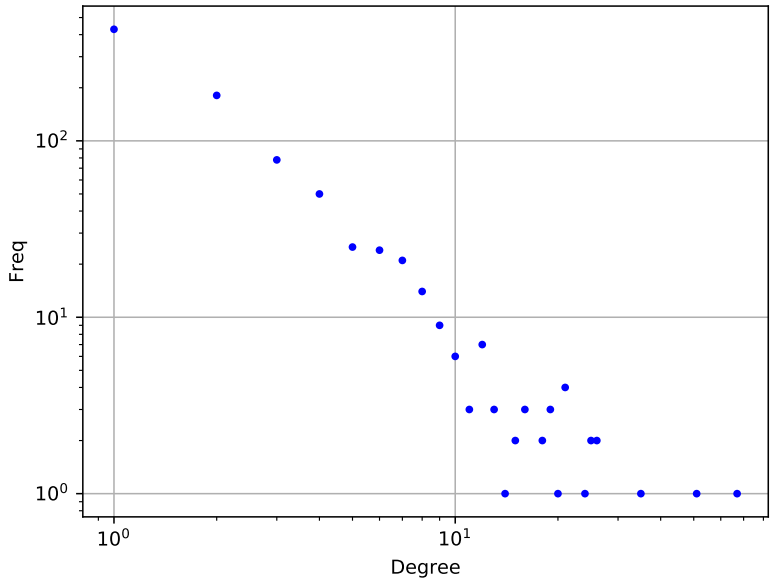


Figure 4.1: Filmtrust degree distribution

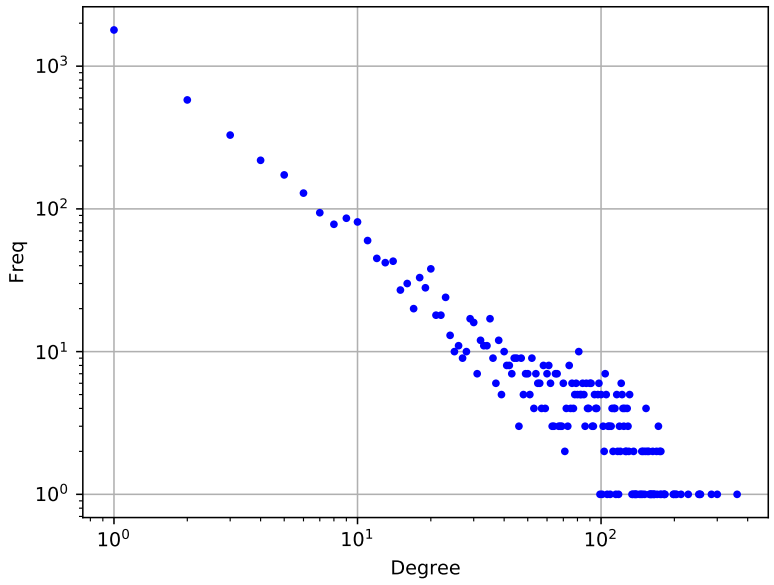


Figure 4.2: CiaoDVD degree distribution

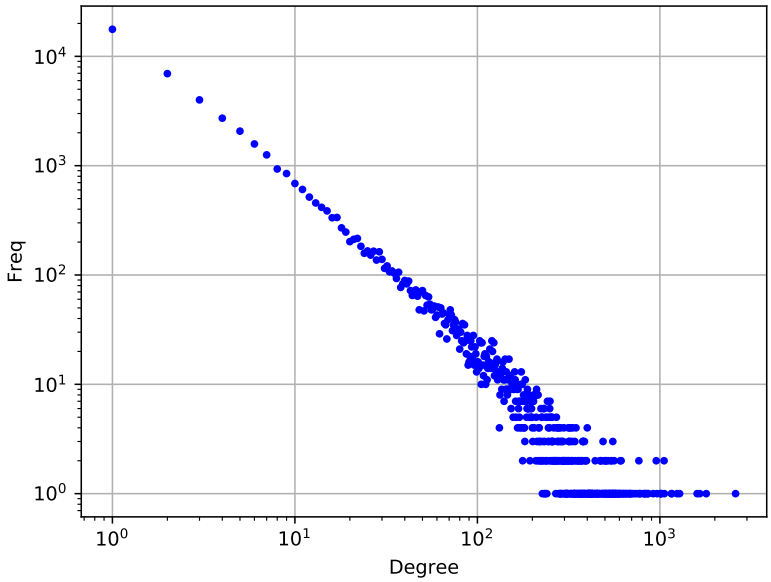


Figure 4.3: Epinions degree distribution

Dataset	User-item interaction			Trust network statistics		
	Users	Items	Ratings	Nodes	Edges	Density
Filmtrust	1,508	2,071	35,497	874	1,853	0.00243
CiaoDVD	17,615	16,121	72,665	4,658	40,133	0.00185
Epinions	49,290	139,738	664,824	49,288	487,183	0.00020

Table 4.1: Statistics of the datasets

4.2 Baselines

The considered algorithms are evaluated on the datasets along with some classic baseline methods from the literature. The following baselines are considered in this work.

Most popular. The most popular items in each dataset, which are considered to be the top 10 frequently rated items, are recommended to all users. This is meant to approximate the mainstream. It can also generate recommendations for any

	All users			Users in the network		
Dataset	Cold-start	Warm	Total	Cold-start	Warm	Total
Filmtrust	545	963	1,508	241	499	740
CiaoDVD	16,591	1,020	17,611	2,124	571	2,695
Epinions	25,393	14,770	40,163	25,393	14,770	40,163

Table 4.2: Statistics of the cold-start users and warm start users who **provided at least one rating** in the datasets

user regardless of their trust statements or their number of ratings, and hence the cold-start and warm users.

Explicit trust. The trust links, outgoing from each user in the network, can be used to find similar users and generate recommendations based on the taste of the user's trustees as a form of simple user-based collaborative filtering. This is a different way of applying collaborative filtering by using the adjacency matrix itself as a similarity matrix, and is rather shallow comparing to the graph embedding based approaches.

Jaccard index. Jaccard index of two sets A and B is generally defined as the intersection over union,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Considering two users u and v with their neighborhoods (sets of trustees) $\mathcal{N}(u) \subseteq V$ and $\mathcal{N}(v) \subseteq V$ respectively, the similarity in this baseline becomes

$$\text{sim}(u,v) = J(\mathcal{N}(u),\mathcal{N}(v)) = \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}, \tag{4.1}$$

and the recommendation is, again, based on collaborative filtering.

Katz similarity. Katz similarity measures the regular equivalence between nodes. It is realized by the recursive intuition that similar nodes are nodes who have similar neighbors, and is mathematically represented as

$$\text{sim} = \sum_{k=0}^{\infty} (\alpha A)^k = (I - \alpha A)^{-1} \tag{4.2}$$

where A is the adjacency matrix and $\alpha \in (0, 1)$ is the attenuation factor. The formula is approximated by setting a limit to k, i.e., k_{max}, and it represents a decaying effect for longer paths comparing to longer paths. After the Katz similarity matrix is computed, there are steps of degree normalization, row normalization and boosting propagated similarities; all explained in depth in [11].

All four baselines were used by [11] for the same task on Epinions dataset.

4.3 Hyperparameter optimization

The graph embedding approaches discussed in Section 3.2 are used with an output dimensionality of 128. In the basic experiments which do not include the directionality study, the direction of the links is ignored to deal with an undirected graph, as preliminary tests on directed graphs showed that the considered methods in general work better with the undirected version of the graph. Next, the used approaches that were used in the experiments are listed with their optimized hyperparameters.

In total, we had 163 combinations for the hyperparameter optimization step, each of which is applied to the three datasets, which gives a total of 489 experiments.

LE and LLE. LE and LLE are not parametrized methods so no settings were tested on them, so there is only **one** test for each of them on each dataset.

Graph Factorization. GF was tested with **three** different regularization weights $\lambda \in \{0.1, 0.5, 1.0\}$.

HOPE. Two different types of high-order proximity were tested for HOPE, i.e., Katz index and Rooted PageRank. Katz index has the decay β parameter which determines how fast the weight of a path decay when the length of path grows as explained in [43]. Rooted PageRank (RPR) depends on a random surfer which, given it starts at a node u, will traverse to a neighbor of the current node with probability α and will jump back to u with probability $1 - \alpha$. Katz index was tested with $\beta \in \{0.01, 0.1, 0.9\}$ and Rooted PageRank was tested with $\alpha \in \{0.5, 0.85, 0.99\}$. This results in **six** different settings for HOPE.

DeepWalk. The main characteristics of DeepWalk are the random walk length, which tells how far each random walk moves from the starting node, and the skip-gram window size, which represents the size of the context to consider while processing each visit in a random walk. The random walk length values considered are in the set $wl \in \{5, 10, 20, 40, 80, 120, 160\}$ and window sizes are in the set $ws \in \{1, 2, 3, 5, 8\}$. In all these experiments, the number of random walks per node is set to 10, In total, we have 35 different combinations for DeepWalk.

node2vec. Similar to DeepWalk, the walk lengths and the window sizes are considered for tuning but with values $wl \in \{20, 40, 80\}$ and $ws \in \{3, 5, 8\}$ respectively. Additionally, the hyperparameter search includes the interesting property of node2vec which is the relationship between the return parameter p and the in-out parameter q. Particularly, for each combination of the walk length and the window size, the chosen cases for p and q are

- p is much greater than q ($p = 1 \gg q = 0.01$),
- p is greater than q ($p = 1 > q = 0.5$),
- p and q are equal ($p = q = 1$),
- q is greater than p ($p = 1 < q = 2$),
- and q is much greater than p ($p = 1 \ll q = 100$).

This sums up to 45 different cases of node2vec.

role2vec. Since role2vec is also random-walk-based, we also consider the walk length as a hyper parameter to optimize with the same values used in DeepWalk: $wl \in \{20, 40, 80\}$. In addition, we optimize the number of clusters that are defined by role2vec as a target of the mapping function ϕ, namely values $\#\text{Clusters} \in \{25, 50, 75\}$, and we try two different types of structural features: Weisfeiler-Lehman (**WL**) features and graph **motif** features with graphlet size of 4. In total, we have 18 different combinations for role2vec.

DNGR. We use **three** different architectures for hidden layers/neurons (without the output layer) of the autoencoder:

- Two hidden layers with sizes 256 and 192: $[256, 192]$
- Two hidden layers both with size 128: $[128, 128]$

- Three hidden layers all with size 128: $[128, 128, 128]$

The first layer has the size of the input, which is the number of the graph nodes, and the last layer carries the output and always has 128 neurons. For each of these architectures, we have a random surfing rate of 0.98 and a random walk length of 10. In total, we have 3 different architectures for DNGR.

GraphSAGE. Four different aggregators are used: mean aggregator, GCN-based aggregator, maxpooling and LSTM-based. For both choices, we try different number of negative samples in $Q \in \{3, 5, 8\}$ and learning rates in $\rho \in \{0.0001, 0.001, 0.01\}$. We run one epoch with model size set to small and sample size of 25 for the first layer and 10 for the second one. We have 36 different combinations for GraphSAGE in total.

LINE. We use the $1^{st}-$ and $2^{nd}-$order proximity and fix the total samples to $100M$. Then we search for the best combination of the number of negative samples in the set $Q \in \{3, 5, 8\}$ and the learning rate in the set

$$\rho \in \{0.0001, 0.001, 0.005, 0.01, 0.025, 0.125\}.$$

This gives a total of 18 parameter settings for LINE.

4.4 Evaluation protocol

For the hyperparameter search phase, we utilize the warm users (validation set) as a target set. We pick the hyperparameter settings that achieve the highest recommendation accuracy (explained bellow) on the warm users. Afterwards, we report our evaluations with the best hyperparameter settings on the cold-start users (test set) and rank the approaches using a statistical ranking test, that is, Friedman test [13]. Some approaches are then considered for the further experiments explained in Section 4.5. We refer to both cases as target users while we explain the evaluation protocol.

4.4.1 Recommendation accuracy

As explained in Section 3.1.3, for a target user (cold-start or warm) with q ratings, we recommend q items after ranking the items based on their score as in Equation 3.1. The accuracy for the recommendation is measured in terms of the nDCG@q [26] metric, the *normalized Discounted Cumulative Gain*, for all target users in the corresponding subset. The nDCG@q is computed as follows after recommending q items to a single user u:

$$\begin{aligned} \text{DCG@}q(u) &= \sum_{i \in REC(u,q)} \frac{2^{rel(u,i)}}{\log(i+1)}, \\ \text{IDCG@}q(u) &= \sum_{i \in REL(u,q)} \frac{2^{rel(u,i)} - 1}{\log(i+1)}, \\ \text{nDCG@}q(u) &= \frac{\text{DCG@}q(u)}{\text{IDCG@}q(u)}, \end{aligned} \tag{4.3}$$

where $rel(u,i)$ is the relevance of the item i for the of the user u, $REC(u,q)$ is the list of q recommended items to u ranked by their score (Equation 3.1) and $REL(u,q)$ is the list of the relevant items to u, i.e., the items which u has rated (these are exactly q in our case) ranked by u's rating. The relevance $rel(u,i)$ is 1 if user u has rated item i and 0 if not.

This score is calculated for each target user and then the average over all target users in the corresponding set is reported for each dataset.

4.4.2 Statistical ranking

To rank the studied embedding algorithms, we apply Friedman test [13] to the methods' nDCG@q score on the three datasets. The ranks of these approaches is then reported with the p-value of the Friedman test.

4.5 Further experiments

4.5.1 Effect of dimensionality

To further compare the representation power of the methods, we pick the best performing parameter setting for each method on each dataset and test it on 9 different output sizes, namely the powers of 2: $\{2^1, ..., 2^9\}$.

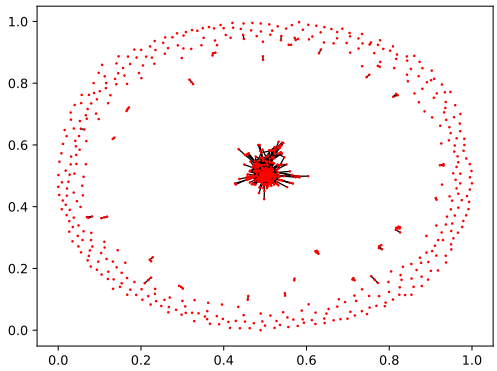
4.5.2 Other versions of the undirected graph

Most of the considered methods are typically not applied to directed graphs in literature [53, 28, 58] as often as they are on undirected graphs. This common behavior has led us to ignore the directions of the links to transform the directed trust network into an undirected one. We further investigate other alternatives, namely the co-citation network and the bibliographic coupling network.

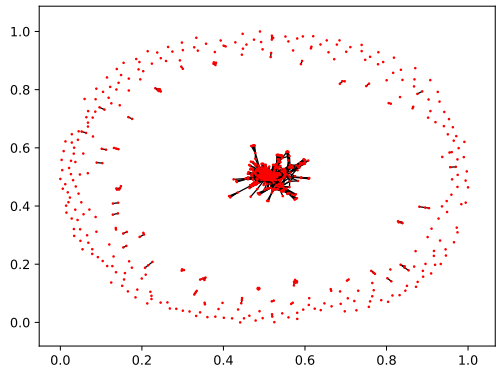
If we consider an unweighted network, represented by a matrix $A \in \mathbb{R}^{n \times n}$ where $A_{i,j}$ represents the edge (i, j), the co-citation matrix $C \in \mathbb{R}^{n \times n}$ where $C_{i,j}$ represents how many times nodes i and j have an outgoing edge to the same node and is calculated by $C = A^T.A$. The bibliographic coupling matrix $B \in \mathbb{R}^{n \times n}$ however, is calculated by $B = A.A^T$ and has $B_{i,j}$ representing how many times nodes i and j have an incoming edge from the same node. Both B and C are symmetric and can be regarded as two undirected weighted graphs. These matrices are not as sparse as A and can practically have a lot of isolated nodes. Table 4.3 shows the statistics of the B and C networks of the three datasets, and Figure 4.4 shows a visualization of Filmtrust network in its undirected version, bibliographic coupling version and in cocitation version.

We take the best performing parameter settings for some methods on each dataset again with an output size of 128 and apply these methods on the matrices B and C of the corresponding dataset. We consider two different alternatives to study the content of B and C and its effect on the embedding.

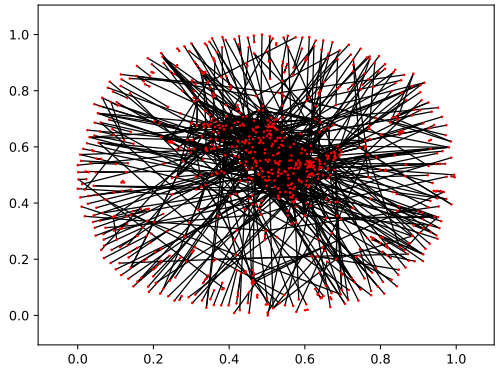
1. Testing each of B and C as they result from the formulas.
2. Ignoring the weights in B and C an dealing with them as unweighted undirected graphs.



(a) Filmtrust bibliographic coupling version visualization



(b) Filmtrust cocitation version visualization



(c) Filmtrust undirected version visualization

Figure 4.4: Filmtrust

For both options, we evaluate each of the two resulting embeddings. These tests only include some of the considered methods in this work because of the high density of the resulting graphs.

Dataset	Co-citation			Bibliographic coupling		
	Edges	Density	Isolates	Edges	Density	Isolates
Filmtrust	11,572	0.01515	256	10,460	0.01369	393
CiaoDVD	957,938	0.04415	393	617,126	0.02844	3,264
Epinions	30,941,630	0.01274	627	39,785,378	0.01638	16,024

Table 4.3: Statistics of the co-citation and the bibliographic coupling networks

5 Results

5.1 Hyperparameter optimization

Following, we present the results of accuracy for the different hyperparameter combinations we chose. Table 5.1 summarizes the parameter search for the considered matrix factorization approaches in terms of accuracy (nDCG@q) on warm-start users subsets. Then we select the best performing settings and apply it to generate recommendations to the cold start users considering them as our test set.

The parameter search for DeepWalk is depicted in Figures 5.1, 5.2 and 5.3 for Filmtrust, Ciao and Epinions respectively as heatmaps of nDCG@q. The parameter search in node2vec and role2vec, is plotted in Figures 5.4 and 5.5 respectively as heatmaps as well.

The heatmap helps recognizing the effect and the sensitivity of each of the hyperparameters. The scale of the color is one for each an approach on a dataset.

The results for different architectures for DNGR are shown in Table 5.2 as there are only 3 choices for a dataset[1].

GraphSAGE, however is again in a heatmap in Figure 5.6 as there are several parameters to explore, similar to LINE which can be seen in Figure 5.7 for its first-order proximity and Figure 5.8 for its second-order proximity.

This optimization ends up by selecting the hyperparameters shown in Table 5.3 for each approach on each dataset.[2]

[1] DNGR was notably very slow as it was not possible to utilize a GPU and it was necessary to have only as few architectures/parameters as possible.

[2] LE and LLE are parameter-less

		Filmtrust	Ciao	Epinions
GF	$\lambda = 0.1$	0.53754	0.01414	**0.01807**
	$\lambda = 0.5$	0.53537	**0.01701**	0.01794
	$\lambda = 1.0$	**0.54257**	0.01412	0.01802
HOPE (Katz)	$\beta = 0.01$	0.54102	**0.01961**	**0.01988**
	$\beta = 0.1$	0.54202	0.01571	0.01181
	$\beta = 0.9$	0.5415	0.0096	0.0085
HOPE (RPR)	$\alpha = 0.5$	0.53995	0.01795	0.01284
	$\alpha = 0.85$	**0.54413**	0.01734	0.01467
	$\alpha = 0.99$	0.53762	0.01791	0.01377

Table 5.1: nDCG@10 score at hyperparameter search for matrix factorization based techniques

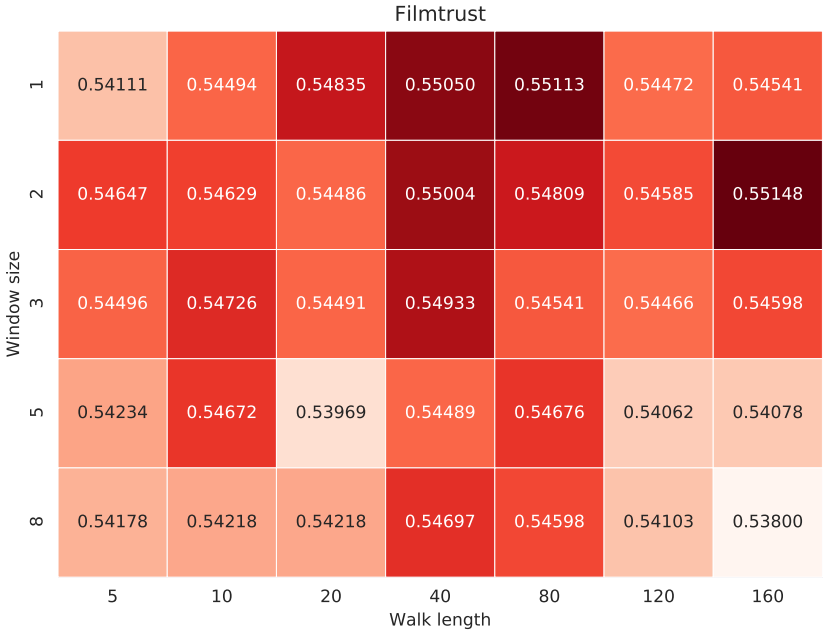


Figure 5.1: DeepWalk hyperparameter effect on Filmturst

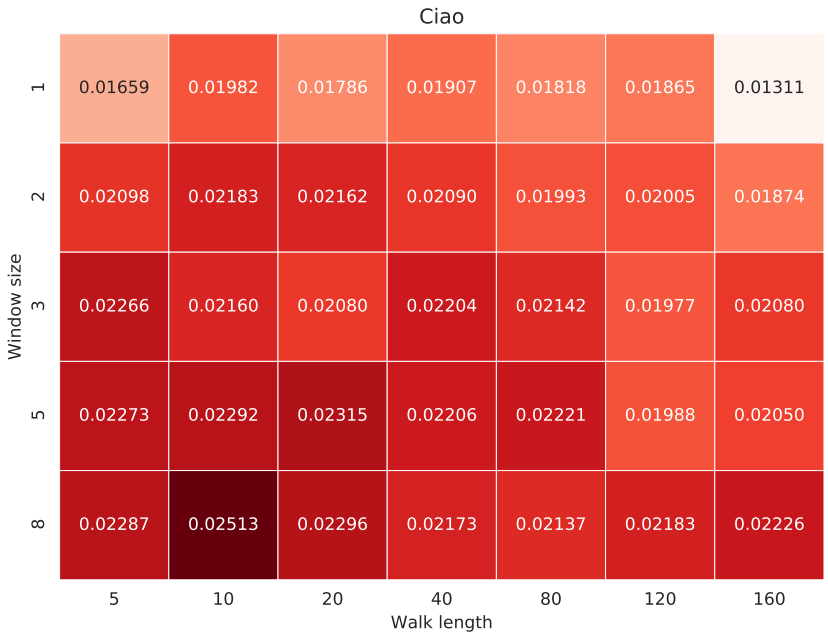


Figure 5.2: DeepWalk hyperparameter effect on Ciao

Hidden Layers	Filmtrust	Ciao	Epinions
[256,192]	**0.54682**	0.01863	**0.02006**
[128,128]	0.54621	0.01861	0.01923
[128,128,128]	0.54607	**0.01994**	0.01918

Table 5.2: nDCG@10 score at hyperparameter search for DNGR

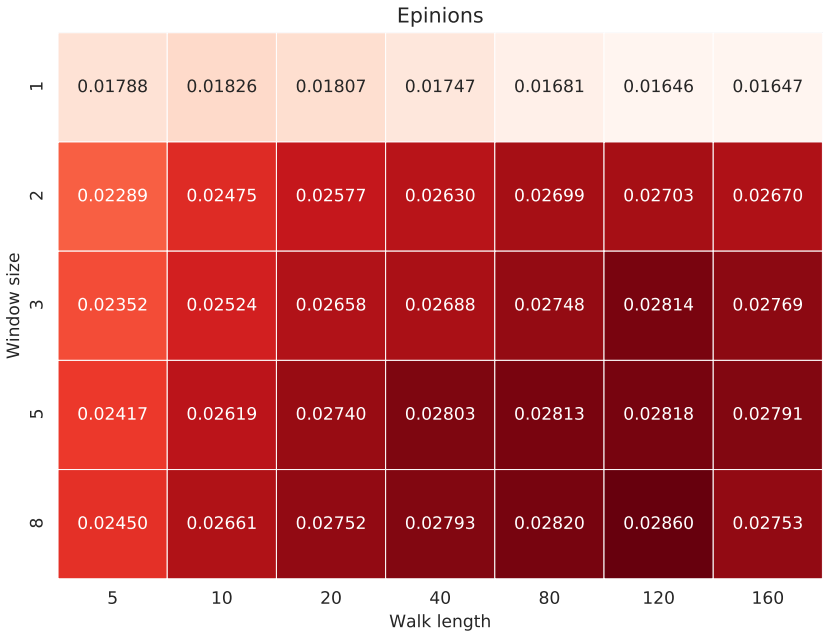


Figure 5.3: DeepWalk hyperparameter effect on Epinions

5.2 Statistical test on recommendation accuracy

Table 5.4 shows the best nDCG@q results for each of the models on the three datasets considering cold-start users. These results are then taken as input for Friedman test that is shown in Table 5.5. The p-value for this Friedman test is 9.3628×10^{-7}.

5.3 Dimensionality

For each of the dataset, after picking the best performing parameter setting for each method in terms of recommendation accuracy (nDCG@q) on the warm-start users subset, we test the method with this parameter setting by having different output sizes (dimensionality).

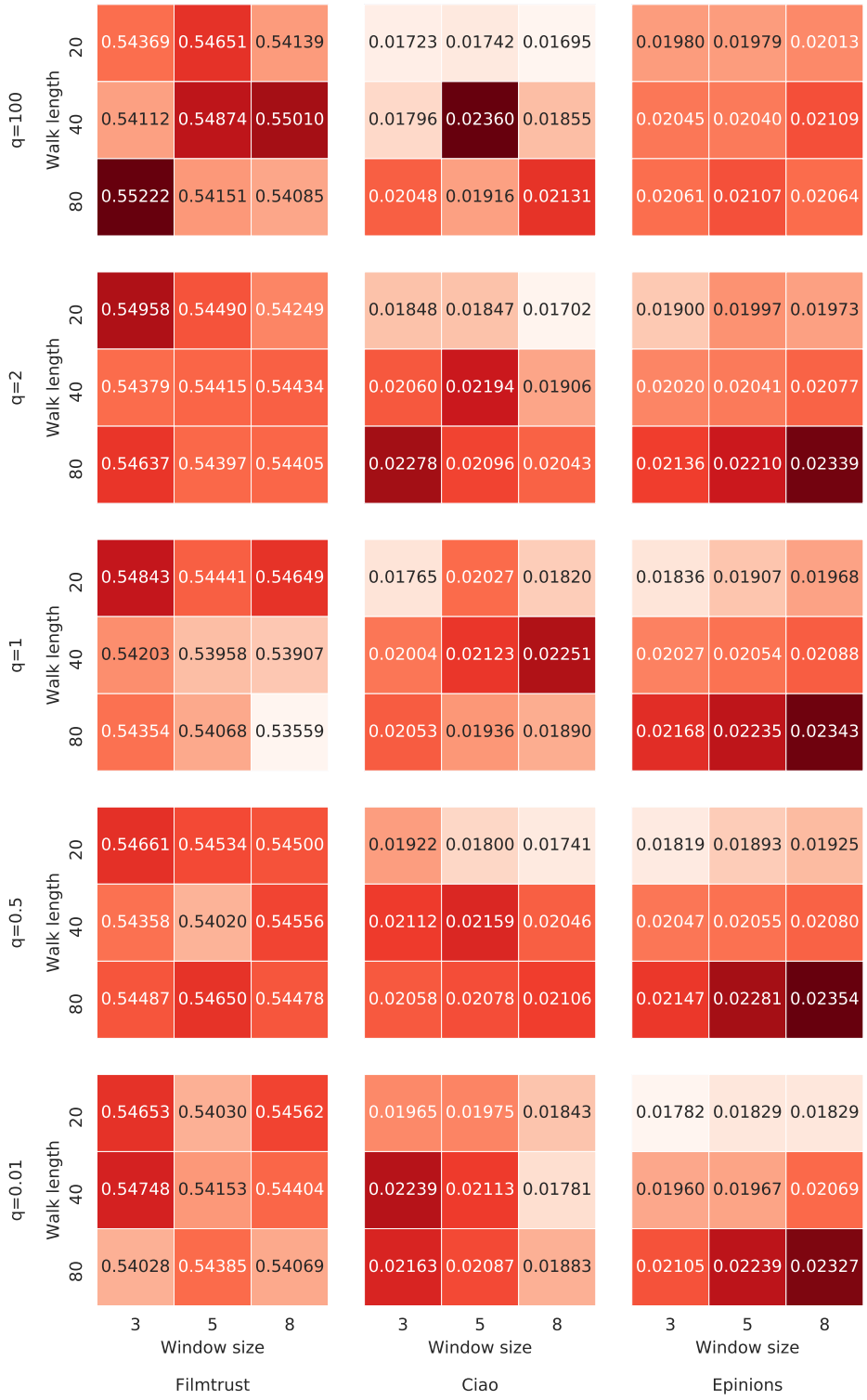


Figure 5.4: node2vec hyperparameter effect

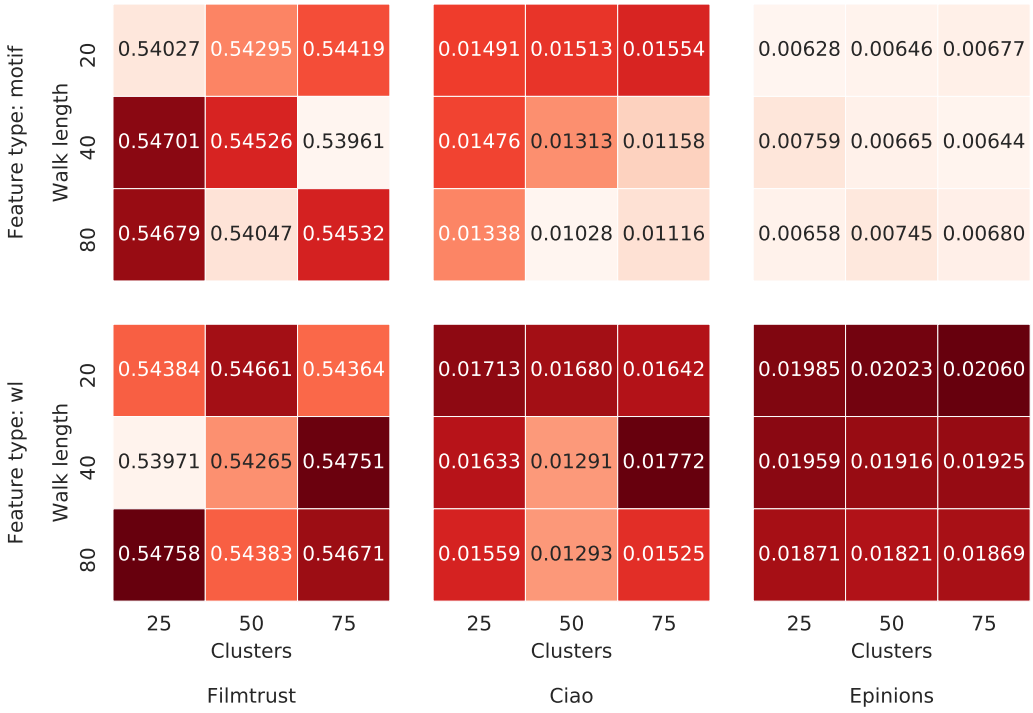


Figure 5.5: role2vec hyperparameter effect

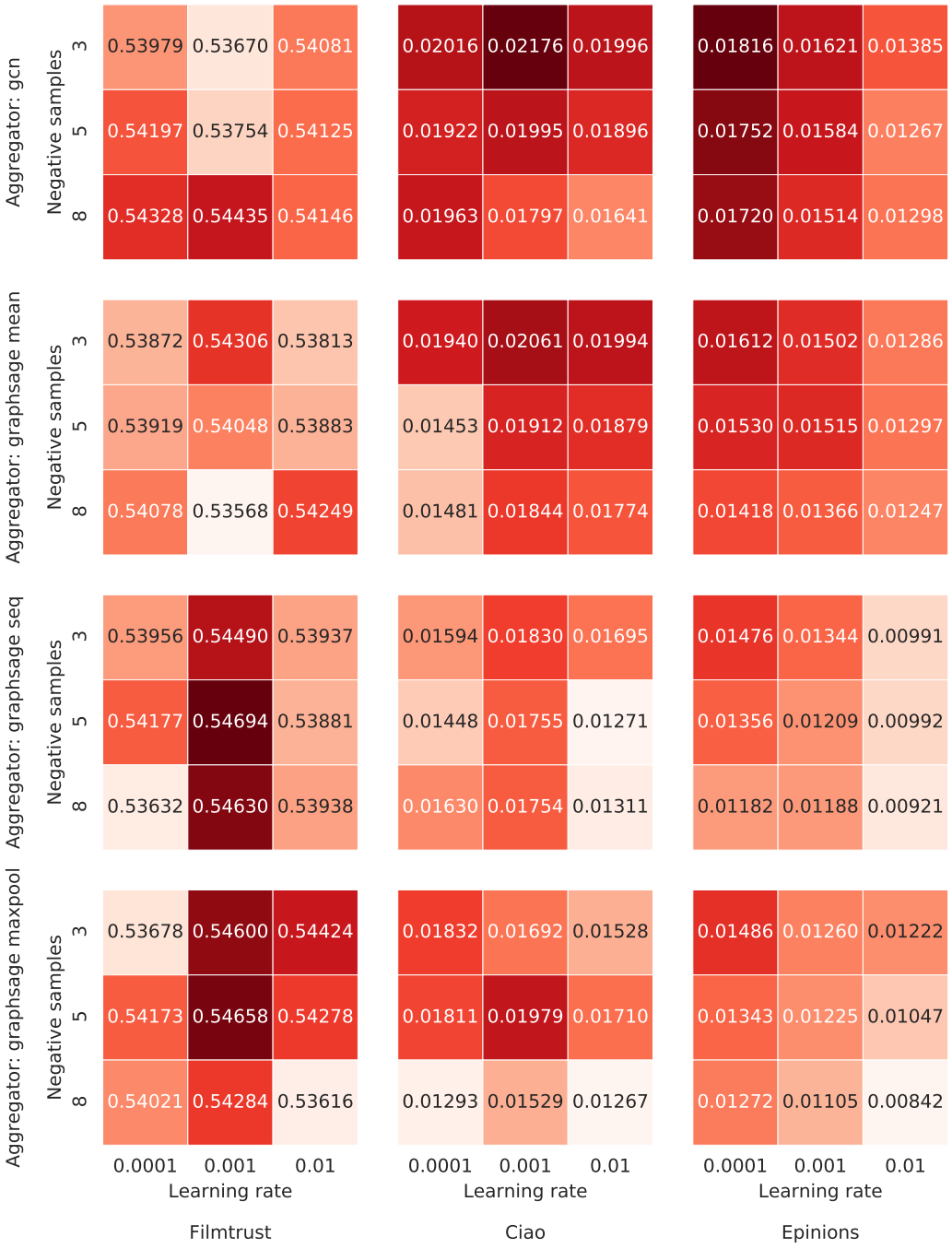


Figure 5.6: GraphSAGE hyperparameter effect

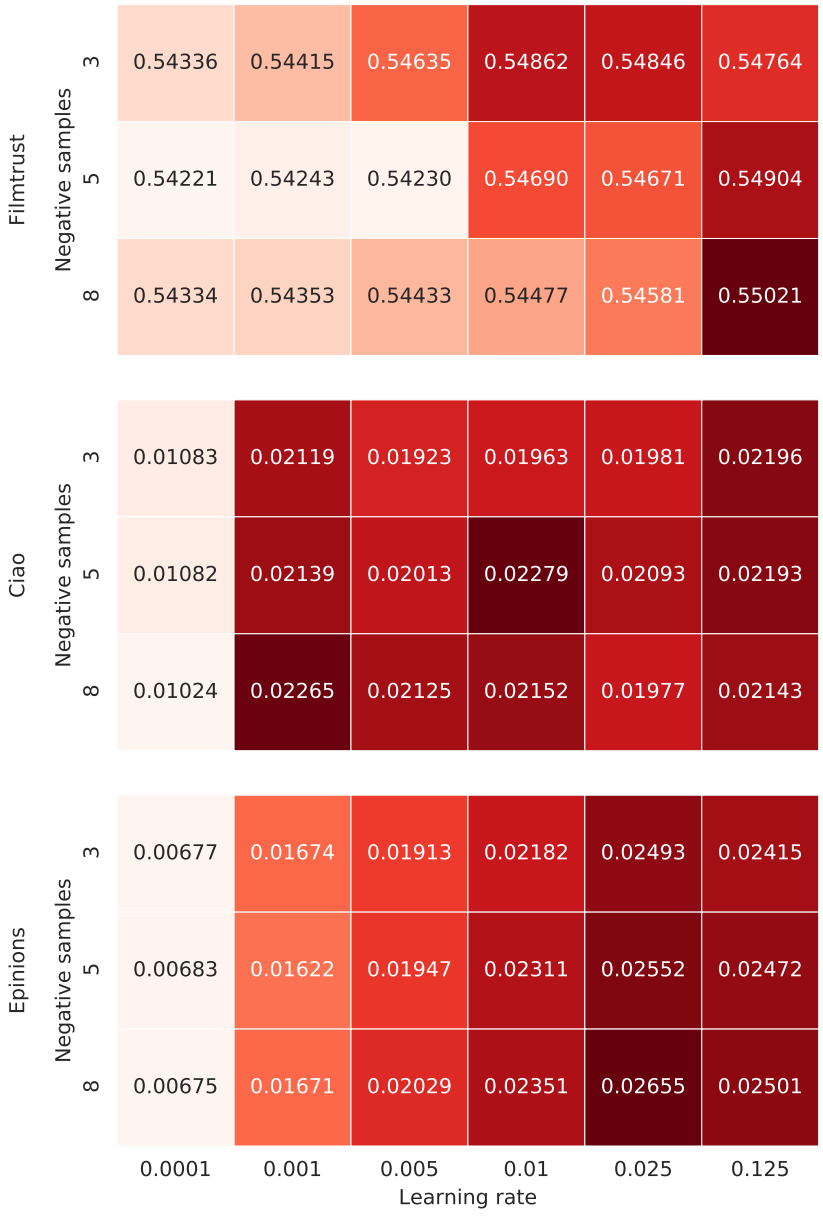


Figure 5.7: LINE hyperparameter effect with 1^{st}-order proximity

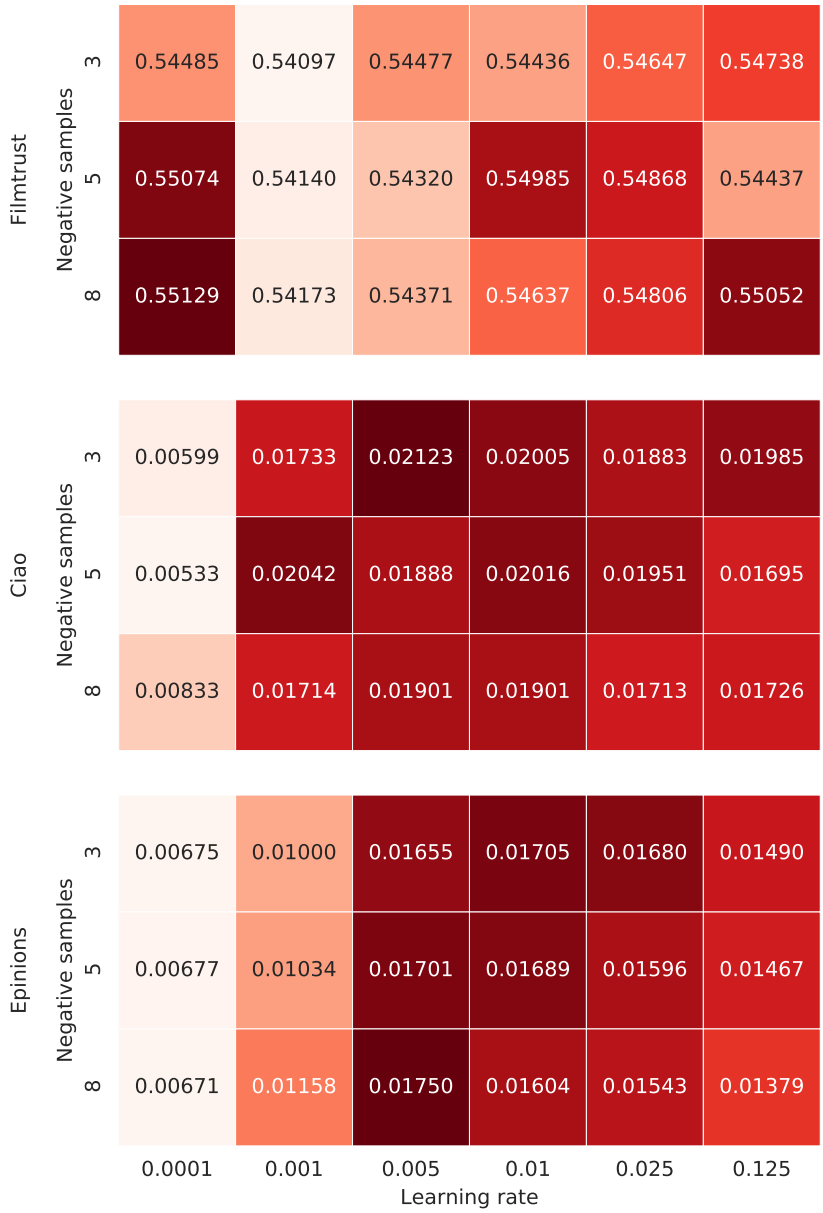


Figure 5.8: LINE hyperparameter effect with 2^{nd}-order proximity

Approach	Parameter	Filmtrust	Ciao	Epinions
GF	λ	1.0	0.5	0.1
HOPE	Similarity	RPR	Katz	Katz
		$\alpha = 0.85$	$\beta = 0.01$	$\beta = 0.01$
Deepwalk	wl	160	10	120
	ws	2	8	8
Node2vec	p	1	1	1
	q	100	100	0.5
	wl	80	40	80
	ws	3	5	8
Role2vec	Feature type	WL	WL	WL
	wl	80	40	20
	# Clusters	25	75	75
DNGR	Hidden layers	$[256, 192]$	$[128, 128, 128]$	$[256, 192]$
GraphSAGE	Model	LSTM	GCN	GCN
	ρ	10^{-3}	10^{-3}	10^{-4}
	Q	5	3	3
LINE	Order	2	1	1
	ρ	10^{-4}	10^{-2}	0.025
	Q	8	5	8

Table 5.3: Best performing hyperparameter setting for each method on nDCG@q of the warm-start users of each dataset.

5.4 Co-citation and bibliographic coupling networks

Table 5.6 summarizes the results of applying each of HOPE, DeepWalk, DNGR and LINE on the unweighted bibliographic coupling networks and co-citation networks of each dataset, whereas Table 5.7 shows the results on the weighted version.

Family	Algorithm	nDCG Filmtrust	Ciao	Epinions
Baselines	Explicit trust	0.2739	0.0132	0.0261
	Most Popular	0.3318	0.0135	0.0134
	Jaccard	0.3387	0.0176	0.0373
	Katz	0.3681	0.0158	0.0290
Factorization Approaches	LLE	0.3649	0.0239	0.0309
	LE	0.3715	0.0231	0.0318
	GF	0.3686	0.0154	0.0138
	HOPE	0.3718	0.0212	0.0331
Random Walk Approaches	node2vec	0.3904	0.0229	0.0413
	DeepWalk	0.3654	0.0247	0.0435
	role2vec	0.3696	0.0138	0.0363
Deep Learning Approaches	DNGR	0.3583	0.0197	0.0353
	GraphSAGE	0.3678	0.0216	0.0325
Other	LINE	0.3667	0.0222	0.0416

Table 5.4: nDCG score of studied methods on the cold-start users of the three datasets after hyper-parameter optimization on the warm-start users.

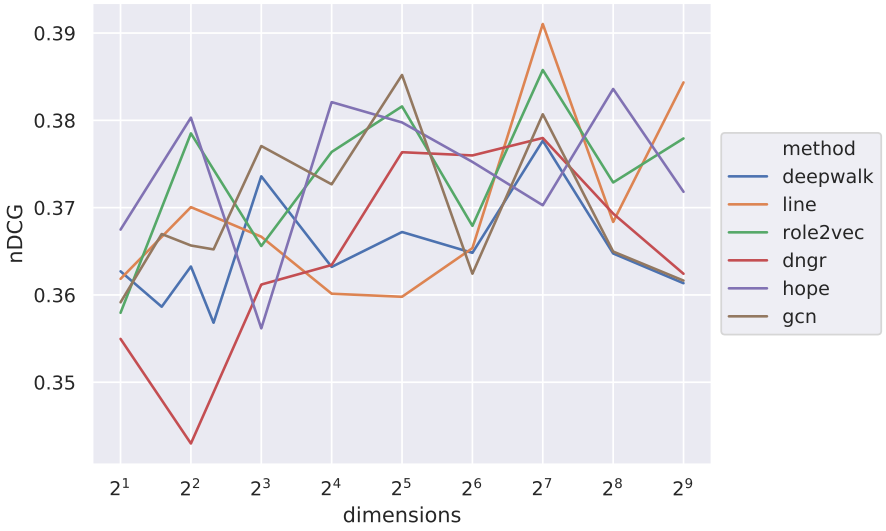


Figure 5.9: Effect of dimensionality through Filmtrust

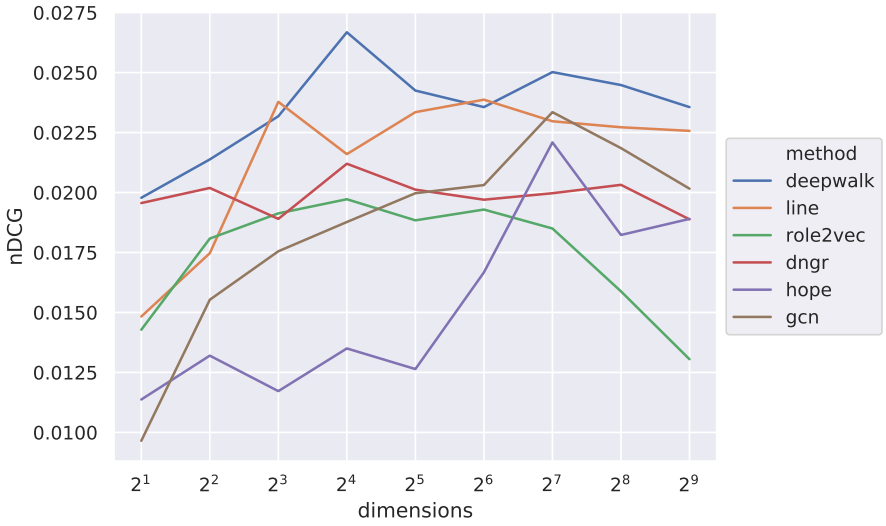


Figure 5.10: Effect of dimensionality through Ciao

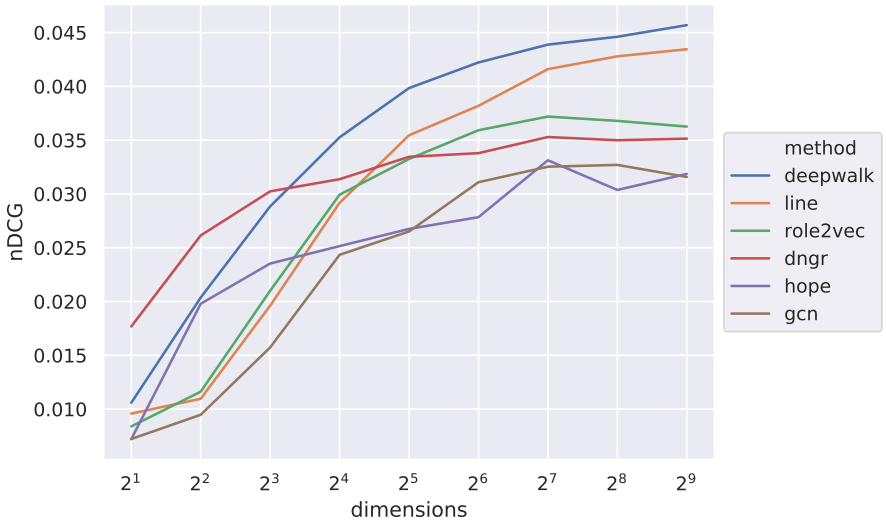


Figure 5.11: Effect of dimensionality through Epinions

		Rank			Rank Sum	Rank by Sum
Family	Algorithm	Filmtrust	Ciao	Epinions		
Baselines	Explicit trust	14	14	12	40	13
	Most Popular	13	13	14	40	13
	Jaccard	12	9	4	25	**9**
	Katz	6	10	11	27	11
Factorization Approaches	LLE	10	2	10	22	8
	LE	3	3	9	15	**3**
	GF	5	11	13	29	12
	HOPE	2	7	7	16	5
Random Walk Approaches	node2vec	1	4	3	8	**1**
	DeepWalk	9	1	1	11	2
	role2vec	4	12	5	21	6
Deep Learning Approaches	DNGR	11	8	6	25	9
	GraphSAGE	7	6	8	21	**6**
Other	LINE	8	5	2	15	3

Table 5.5: Friedman test ranks on the obtained nDCG results for all considered methods.

	Filmtrust		Ciao		Epinions	
	B	C	B	C	B	C
HOPE	0.32755	0.36238	0.01048	0.02262	0.00817	0.02312
DeepWalk	0.3364	0.34573	0.00715	0.01877	0.0082	0.02613
DNGR	0.34782	0.35213	0.00949	0.0228	0.00777	0.02359
LINE	0.34338	0.36081	0.00776	0.01932	0.00734	0.01931

Table 5.6: nDCG@10 score for unweighted bibliographic coupling network and cocitation network

	Filmtrust		Ciao		Epinions	
	B	C	B	C	B	C
HOPE	0.32755	0.35715	0.01095	0.02262	0.00811	0.02312
DeepWalk	0.33446	0.36233	0.00732	0.02093	0.00779	0.02561
DNGR	0.35176	0.3506	0.00604	0.02411	0.00817	0.03346
LINE	0.35397	0.35633	0.01318	0.01859	0.00721	0.01745

Table 5.7: nDCG@10 score for weighted bibliographic coupling network and cocitation network

6 Discussion

6.1 Statistical ranking of nDCG scores

The ranking in Table 5.5 shows that all the graph embedding approaches achieve higher recommendation accuracy (nDCG) than the baselines with the exception of Graph Factorization (GF) which is outperformed by Jaccard index and Katz similarity baselines. Despite its simplicity, the Jaccard index baseline achieves a competitive accuracy on Epinions.

node2vec has the overall highest rank, with the overall best accuracy on Filmtrust, which also makes it the best performing random-walk-based approach considered in this work. Random-walk-based approaches achieve ranks 1 , 2 and 6 (for node2vec, DeepWalk and role2vec respectively) making them the family with the highest average rank. Althoug DeepWalk outperforms all others on Ciao and Epinions, it has a rather low accuracy on Filmtrust such that it is not ranked first, but second overall. LINE achieves the third highest ranking overall tied with LE which achieves a competitive accuracy on Filmtrust and Ciao, the smaller datasets. LE, thus, achieves the highest rank among the factorization-based approaches surprisingly beating HOPE which is a more complex approach. GraphSAGE ranks the highest between the two considered deep-learning-based approaches with the rank 6 overall. This underwhelming performance of the deep learning approaches highlights that such approaches may be unable to scale to unsupervised learning on graphs, especially with the target loss functions they have.

6.2 Hyperparameter sensitivity

In this section, we mention the interesting observations about the performance of the examined approaches. We exclude LLE and LE as they are parameter-less, and

GF as the only tested parameter is the learning rate.

HOPE. Using Rooted PageRank as a high-order similarity seems to achieve a higher accuracy on Filmtrust whereas using Katz index outperforms it on Ciao and Epinions in average. Katz index shows a diverse performance with changing attenuation factor β on Ciao and Epinions, and also shows a significantly higher performance with a very small attenuation factor.

DeepWalk. An interesting hyperparameter to analyse in DeepWalk is the random walk length. Figure 6.1 shows, for each dataset, the average nDCG score of each choice of random walk length on cold-start users, taking different window sizes (normalized for comparison purposes). It shows a decrease in accuracy on Filmtrust as the walk length increases. Meanwhile, it shows an increase in accuracy on Epinions with the increasing walk length, that converges starting the walk length of 80. The accuracy on Ciao, however, drops at the lengths of 20 and 40 and then remains steady starting 80 although it obtains its best average accuracy for a walk length of 10. These observations support the hypothesis that longer random walks suit larger graphs and vice versa.

node2vec. The effect of the relative relation between the return parameter p and the in-out parameter q on nDCG reflects the dependence of this score on the homophily and the structural role of the nodes in the considered datasets.

Figure 6.2 shows the average nDCG (max-min-normalized for comparison) for all runs with different walk lengths and window sizes on each choice of p and q[1] for each dataset. Although the maximum nDCG was realized using lower q for Epinions, all three datasets get the highest average nDCG for higher q with Epinions and Filmtrust tending to the case where $p \ll q = 100$.

Figure 6.2 indicates that utilizing structural roles of nodes leads to higher accuracy **on average**. Nevertheless, the bigger range of nDCG that using lower q for Epinions (Figure 5.4) indicates a higher response to changes in random walk length and window size when using smaller q (homophily). This is reflected in Figure 6.3 where

[1] p is always 1 while q changes $\in \{0.01, 0.5, 1, 2, 100\}$

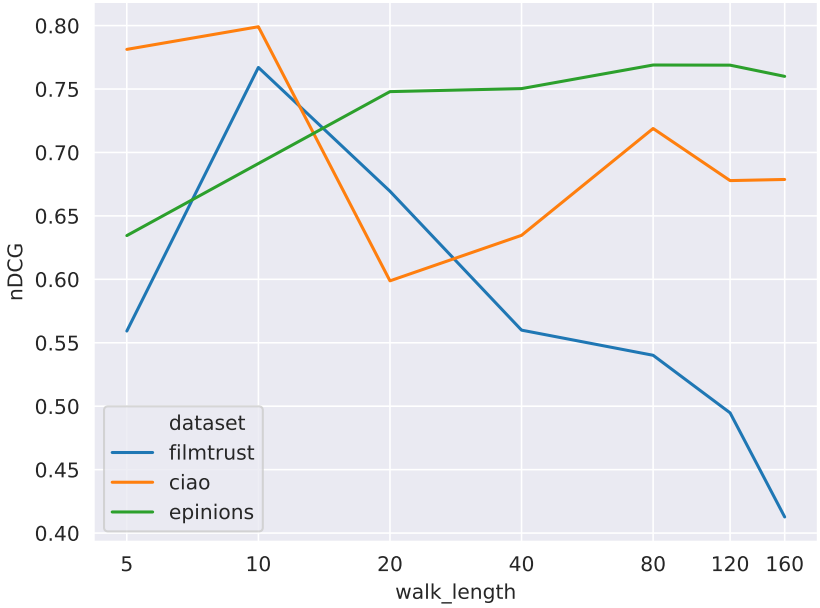


Figure 6.1: Effect of walk length on the performance of DeepWalk. For each dataset, the nDCG scores are normalized via min-max normalization and then the average of these scores per walk length choice is reported.

the standard deviation of Epinions drops dramatically as q grows. Ciao also shows a higher nDCG standard deviation for lower q while Filmtrust does not.

This suggests that lower q gives a higher variance as the walk length and the window size changes, and recommends to optimize these two parameters in case lower q is selected. Such an phenomenon can be interpreted as a consequence of the oversampling performed when p is lower, that is, a walk tends to cover a small area in the network as it has more probability to return and hence the random walk length and the window size would have less of an effect. On the other hand, a lower q indicates deeper walk, which can be more responsive to changes in walk length and window sizes as they discover broader regions in the network.

role2vec. From Figure 5.5, the graph Weisfeiler-Lehman features increased the performance on all considered datasets, whereas the poor performance of motif features was mostly obvious on Epinions. The other two hyperparameters did not

Figure 6.2: Mean nDCG on the cold start users for node2vec's q choices averaged over walk lengths and window sizes to reflect the effect of p and q values on node2vec accuracy. $p = 1$ for all the cases of q. For each dataset, the nDCG scores are normalized via min-max normalization and then the average of these scores per walk length choice is reported. NOTE: THE SCALE OF AXIS q IS TRANSFORMED.

show a certain interesting pattern on any of the datasets.

DNGR. Following Table 5.2, the architecture with more neurons in one layer was the most suitable for Epinions and Filmtrust, while the deeper architecture was the most suitable for Ciao.

GraphSAGE. Figure 5.6 shows that the best model type for GraphSAGE on two datasets (Ciao and Epinions) was the GCN model. Despite the representational power that the LSTMs have, GraphSAGE performed quite poorly with the LSTM-based aggregator. On the other hand, the simpler models, that are GCN-based and mean-based aggregators, obtained a better score. The case of Epinions also shows that less negative samples and smaller learning rate achieve a higher score

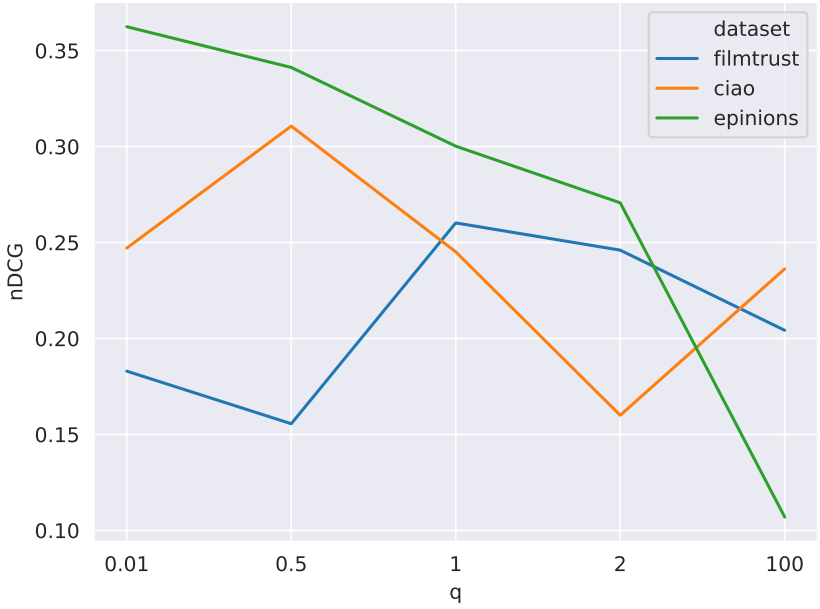


Figure 6.3: Standard deviation of nDCG on the cold start users for node2vec (grouped similar to Figure 6.2). For each dataset, the nDCG scores are normalized via min-max normalization and then the average of these scores per walk length choice is reported. NOTE: THE SCALE OF AXIS q IS TRANSFORMED.

whereas this was not the case for Ciao which mostly optimized with a learning rate $\rho = 0.01$.

LINE. First-order proximity outperformed the second-order one on Ciao and Epinions (Figures 5.7 and 5.8). The learning rate seems to have a higher impact than the number of negative samples on both Ciao and Epinions and using both orders of proximity.

6.3 Dimensionality

Looking at Figure 5.9, one can notice an obvious oscillation in Filmtrust case with a slightly noticeable overall increase as the number of dimensions increases.

Moreover, a jump in accuracy is also obvious for 128 dimensions. This last observation highlights that the choice of hyperparameters for Filmtrust was not an indicator of their suitability for this dataset.

To examine the oscillation, a Friedman test was again applied to the measurements with different dimensions on each dataset for the six approaches. The resulting p-value was rather high for Filmtrust p-value$= 6.3 \times 10^{-2}$ comparing to Epinions p-value$= 8.71 \times 10^{-6}$ and Ciao p-value$= 1.27 \times 10^{-5}$. This high p-value in case of Filmtrust shows another indication of the insignificance of the differences achieved via different dimensions on this dataset.

Figure 5.10 shows more interpretable patterns for Ciao with most approaches starting with a low accuracy and increasing gradually with more dimensions. While DNGR seems to saturate at an early stage and remains steady afterwards, most other methods start to drop at some point, most notably with role2vec which drops after increasing the number of dimensions to more than 64.

Finally, in Epinions case (Figure 5.11), all methods increase in nDCG with more dimensions. While DeepWalk outperforms the others starting 64 dimensions, DNGR starts strongly with a low number of dimensions getting the highest nDCG among all. Most methods converge as early as 64 dimensions whereas LINE and DeepWalk do not show an obvious convergence in this range and get higher nDCG score for higher dimensions.

6.4 Directionality

Using the bibliographic coupling and cocitation graphs was not helpful. Looking at Tables 5.6 and 5.7, no increase in accuracy was recorded over the classical undirected approach, that is ignoring link directions. Moreover, the processing of such graphs is much more resource-consuming as the graphs are denser (Table 4.3).

Notably, the cocitation case constantly outperformed the bibliographic coupling case. This is a possible consequence of having many more isolates in bibliographic coupling versions than in the cocication versions for all of our considered graphs.

To leverage the directionality information in the graph, we perform further experiments in a parallel work [2] where we evaluate different methods (including HOPE, LINE, node2vec, GCN and GraphSAGE) on the semi-supervised classification problem using three benchmark citation datasets: Cora, CiteSeer and Pubmed [49]. The input to the compared methods is the graph in three different cases: undirected, directed and reverse-directed. The results from that work (see Table 6.1) support the idea that ignoring the link direction is a feasible solution which results in an acceptable accuracy (for semi-supervised classification in that case and for recommendation accuracy in our case).

However, we propose an extension to common graph convolutional networks

Table 6.1: *Top:* Classification accuracy on directed graphs: (cited → citing), (cited ← citing), and undirected version (cited ↔ citing). The baseline is a fully connected neural network with 16 neurons in one layer. (*) hub representation, (**) authority representation. LINE(1) (only applicable for undirected graphs) and LINE(2) refer to LINE with 1^{st}- and 2^{nd}-order proximity respectively. *Bottom:* Comparison of the three different convolutional architectures. The best accuracy for a type and a dataset is underlined, the best accuracy overall for a dataset is additionally in bold.

Dataset	Cora			CiteSeer			Pubmed		
Baseline	55.81			61.23			61.27		
Directionality	↔	→	←	↔	→	←	↔	→	←
HOPE [43]	64.00	59.50	59.50	61.40	61.20	61.20	70.00	68.20	68.20
LINE(1) [52]	65.80	—	—	63.90	—	—	54.60	—	—
LINE(2) [52]	67.00	57.20	58.70	66.40	56.90	62.30	65.20	49.30	58.20
node2vec [18]	76.70	57.10	56.00	67.70	61.70	30.00	70.60	39.40	54.20
GCN [28]	80.93	63.68	70.26	72.12	62.90	65.41	77.34	66.80	65.75
SAGE [21]	80.45	66.75	73.20	72.14	62.29	68.21	77.07	69.21	68.16
GAT [53]	80.91	66.73	73.36	73.64	65.32	70.31	77.62	68.34	67.52
Architecture	↔	AST	SST	↔	AST	SST	↔	AST	SST
GCN	80.93	80.49	80.21	72.12	71.89	71.79	77.34	77.21	77.16
SAGE	80.45	80.05	79.79	72.14	73.00	72.73	77.07	77.64	**77.83**
GAT	80.91	81.04	**81.33**	**73.64**	73.18	73.33	77.62	75.45	76.26

[2] The work is in a preliminary stage and being prepared for submission to a relevant venue.

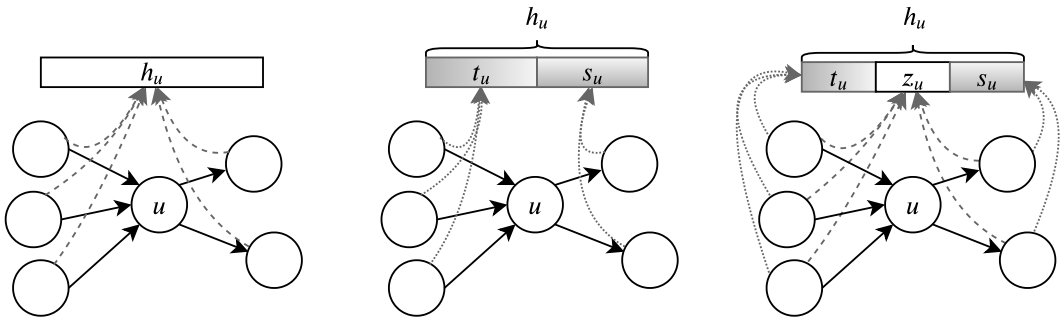


Figure 6.4: *Left:* Classical undirected propagation architecture in graph convolution. *Center:* Proposed asymmetric source-target (AST) extension that differentiates between incoming and outgoing links while performing the convolution. *Right:* Proposed symmetric source-target (SST) architecture allowing symmetric information flow to z_u in addition to the differentiation in AST. Solid lines represent directed edges, dashed lines represent the symmetric flow, and dotted lines represent the asymmetric flow.

architecture. The extension, explained in Figure 6.4, incorporates the directions information in a simple manner, that is, differentiating between information propagated from incoming edges and information propagated from outgoing edges.

The results of applying these extensions to known graph convolutional models, e.g. GCN, GraphSAGE and GAT, shows a slight improvement in accuracy in some cases (namely, GAT with SST and AST on Cora as well as SAGE with AST on CiteSeer and with SST on Pubmed) while still shows inferior accuracy in others. The interpretation we propose to this phenomenon is the dependence of the labels in graphs of such nature on community structure, i.e., nodes in the same community tend to have similar labels. To verify this, we aim to destroy community structure in the given networks while keeping the degree distribution and compare the convolutional architectures on the resulting graph in the three different versions: undirected, directed and reverse-directed.

We apply the configuration model [41] on the given datasets, run the methods and report the accuracy again. The results (Table 6.2) show that the accuracy for the three different graph versions on the different architectures sink to a similar point, which suggests that destroying the community structure destroys the ability of recognizing the labels for all considered propagation topologies. This leads to assume that community structure has the most valuable information graphs of similar nature on the semi-supervised classification task.

Table 6.2: The accuracy of convolutional methods on the configuration model of the considered datasets.

Dataset	Convolution type	↔	AST	SST	→	←
Cora	GCN	31.45	30.88	30.79	29.26	31.61
	SAGE	26.50	30.42	31.01	20.34	26.74
	GAT	22.74	27.51	27.36	19.71	30.52
CiteSeer	GCN	35.33	34.23	34.01	32.69	34.90
	SAGE	30.02	34.03	34.55	26.85	26.58
	GAT	35.97	35.04	33.76	28.40	31.67
Pubmed	GCN	52.21	52.28	52.23	55.89	59.72
	SAGE	48.51	48.38	48.36	41.24	59.26
	GAT	49.23	54.40	54.90	42.83	58.44

7 Conclusion and future work

This work compares 10 different graph embedding approaches from four graph embedding families on the trust-based recommendation task for cold-start users. We show that the random-walk based approaches are a good choice for this problem with node2vec achieving the highest accuracy. Deep-learning based approaches were notably weak, an observation that implies a low-quality representation for the unsupervised embedding on graphs. The evaluations of different random walk lengths supported the hypothesis that larger graphs need longer random walks while long walks do not perform well on smaller graphs. Experiments on node2vec showed that a stronger response for random walk length and context size is realized by utilizing homophily instead of nodes structural roles for the given datasets.

We outline the impact of the number of output dimensions that has to be optimized carefully. Moreover, we show that ignoring the link directions is a feasible, efficient, effective solution to deal with directionality in comparison to using the bibliographic coupling network or the co-citation network. Furthermore, we try to interpret the low performance when ignoring the link direction as a consequence of a natural correlation between community structure and the nodes properties.

Future work can explore the performance of different graph embedding families and approaches on different tasks. Moreover, the scalability of deep-learning-based approaches to unsupervised learning on graphs can be further investigated through new methods of defining a target loss function. Finally, other solutions and models to exploit directionality information in graphs can be a direction of interest.

Bibliography

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 2001.

[2] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[3] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry. Learning role-based graph embeddings. *arXiv preprint arXiv:1802.02896*, 2018.

[4] R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, and M. Tennenholtz. Trust-based recommendation systems: an axiomatic approach. In *Proceedings of the 17th international conference on World Wide Web*, pages 199–208. ACM, 2008.

[5] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[7] H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[8] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900. ACM, 2015.

[9] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[10] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[11] T. Duricic, E. Lacic, D. Kowald, and E. Lex. Trust-based collaborative filtering: Tackling the cold start problem using regular equivalence. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 446–450. ACM, 2018.

[12] T. Duricic, E. Lacic, D. Kowald, and E. Lex. Exploiting weak ties in trust-based recommender systems using regular equivalence. *arXiv preprint arXiv:1907.11620*, 2019.

[13] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[14] H. Gao, Z. Wang, and S. Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1416–1424. ACM, 2018.

[15] J. Golbeck, J. Hendler, et al. Filmtrust: Movie recommendations using trust in web-based social networks. In *Proceedings of the IEEE Consumer communications and networking conference*, volume 96, pages 282–286. Citeseer, 2006.

[16] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[17] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[18] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[19] G. Guo, J. Zhang, D. Thalmann, and N. Yorke-Smith. Etaf: An extended trust antecedents framework for trust prediction. In *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 540–547. IEEE Press, 2014.

[20] G. Guo, J. Zhang, and N. Yorke-Smith. A novel bayesian similarity measure for recommender systems. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[21] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[22] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[23] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[24] D. Helic. Lecture notes in network science, October 2017.

[25] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[26] K. Järvelin, S. L. Price, L. M. L. Delcambre, and M. L. Nielsen. Discounted cumulated gain based evaluation of multiple-query ir sessions. In C. Macdonald, I. Ounis, V. Plachouras, I. Ruthven, and R. W. White, editors, *Advances in Information Retrieval*, pages 4–15, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[27] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[28] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[29] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[30] H. Li, D. Wu, W. Tang, and N. Mamoulis. Overlapping community regularization for rating prediction in social recommender systems. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 27–34. ACM, 2015.

[31] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[32] L. Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993.

[33] H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 931–940. ACM, 2008.

[34] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[35] P. Massa and P. Avesani. Trust-aware collaborative filtering for recommender systems. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 492–508. Springer, 2004.

[36] P. Massa and P. Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[37] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[38] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[39] A. Mnih and G. E. Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[40] F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[41] M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[42] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[43] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.

[44] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

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

[46] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

[47] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[48] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[49] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[50] Y.-D. Seo, Y.-G. Kim, E. Lee, and D.-K. Baik. Personalized recommender system based on friendship strength in social network services. *Expert Systems with Applications*, 69:135–148, 2017.

[51] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 322–335. ACM, 2009.

[52] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[53] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[54] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[55] W. Woess. Random walks on infinite graphs and groups—a survey on selected topics. *Bulletin of the London Mathematical Society*, 26(1):1–60, 1994.

[56] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[57] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[58] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[59] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 2018.