Master Thesis

Machine Intelligence

Learning Network Features: Link Prediction Framework for Bigraphs

Author:
Philip Thruesen

Supervisor:
Dr. Peter Dolog

Department of Computer Science
Aalborg University
Denmark

June 2, 2017

1. Abstract

Prediction of links in networks often requires an effort in feature engineering and utilization of expert knowledge. Much of today's research are primarily domain-specific, simply tuning variations of basic methods to fit a particular purpose.

Network embedding is a method to learn low-dimensional representations of vertices in a network, and recently such methods have been significantly improved by borrowing from advancements in the field of natural language processing.

We propose to apply feature learning on topological network data and thereby learn network representations specifically for link prediction in bipartite networks. We present a novel algorithmic framework for exploiting recent methods of network embedding onto bipartite networks with the purpose of predicting links. Applications of our framework may be in any field having bipartite network representations.

We demonstrate improvements over state-of-the-art techniques for link prediction. Besides achieving better precision, our proposed framework has further advantages including high modularity, scalability and a by-product in the form of vertex embeddings, useful for other unrelated machine learning tasks.

2. Introduction

A bipartite network is a specific class of network consisting of 2 partitions of vertices where every edge has an end in each of the partitions [1]. This particular type of network is suitable for describing pair-wise relations between entities e.g. users and items [2]. The networks are applicable to many real-world scenarios including the modeling of metabolic network of chemicals and reactions [3], diseases and genes [4] and ingredients (food) and flavors [5]. Without loss of generality we will mainly focus on bipartite networks used in social, economic, and information systems. Such networks may contain information on preferences, opinions and collaboration, commonly present in the fields of information retrieval and recommender systems [6].

Working with real-world network structures we are often challenged by the sparse and high dimensional data. Bipartite networks (also called bigraphs) contain more information than its unipartite projections [7], though the curse of dimensionality is even greater.

We propose an algorithmic framework using semi-supervised dimensionality reduction methods for the purpose of link prediction in bipartite networks.

Applications of our framework may be in most fields where bipartite network representations are appropriate, including in information retrieval, recommender system and more.

We suggest to employ network embedding algorithms for dimensionality reduction as part in our framework due to the many recent advancements in the field [8] [9] [10] [11] [12]. These non-linear methods construct low-dimensional continuous vector embeddings of vertices in a network. An essential property of such vector is that if 2 vectors are near each other in the embedding space, their corresponding vertices are (by some definition) closer related. This is the network equivalent of the Skip-Gram model [13] for natural language processing where words appearing in similar contexts (e.g. the same sentences) have corresponding vector representations that are near in the embedding space.

In our proposed framework we build a family of networks based on the input of a single weighted bipartite network. We utilize state-of-the-art network embedding algorithms for constructing low-dimensional vector representations of entities (represented as vertices). These vectors are used for constructing a set of numerical features optimized for link prediction using linear regression. The outcome is a score proportional to the likelihood of link existence between pairs of unconnected vertices, which is the common approach for most link prediction methods [14]. For example, in some recommender systems a user might see it fit to retrieve an ordered list of items he/she is most likely to engage with.

The proposed framework outperforms current state-of-the-art matrix factorization techniques for link prediction. Furthermore, our framework has a natural modularity allowing replacements and extensions of task-specific algorithms such as the method for network embedding, which may be relevant as future research advances in the respective field.

Another notable benefit of our framework is the potential multipurpose usage of learned vertex representations for other downstream machine learning tasks. This includes multilabel classification of vertices which is the main focus in most network embedding articles referenced by this paper.

Our contributions are as follows:

- We introduce a novel algorithmic framework as a

tool for predicting links in bipartite networks.

- We perform thorough experiments on our proposed framework and compare with baselines methods including current state-of-the-art.
- We compare leading techniques in performing network embedding for the purpose of downstream link prediction.
- We demonstrate our model on 3 real world datasets.

To the best of our knowledge, we are the first to employ current state-of-the-art network embedding methods for link prediction in bipartite networks.

The rest of the paper is structured as follows. In Section 3, we discuss related work. In Section 4, the problems and tasks are defined, first conceptually, then formally. Section 5 contains a detailed description on our solution to identified problems and their theoretical properties. We perform thorough experiments in Section 6 with various evaluation metrics, baseline comparisons, visualization, and optimization of hyperparameters. Finally, we summarize and conclude in Section 7 and 8.

3. Related Work

In this section, we will discuss work related to our objective of link prediction, and furthermore work related to our most essential framework building block, network embedding.

Link prediction is a problem generally known for its application in social networks, though the problem also has applications in other domains like information retrieval, bioinformatics and e-commerce [15].

A recent survey by V. Martínez et al. [14] mentions link prediction as an open and relatively young field of research. They emphasize that searching for which network structural properties leads to better link prediction is among the most important issues yet to be disclosed. In their survey, they observe that most of todays research in link prediction is domain specific, simply being tuned variations of basic methods. In line with their work, we are more focused on a topological-based link prediction which serves more domains that attribute-based models. V. Martínez et al. conclude that the main challenge ahead is to develop scalable models that utilize more than just local network information while still being applicable on large real-world dataset.

The recent emergence of high performance, accurate and scalable network embedding techniques allows for improvements in link prediction problems as many such techniques strike a balance between local and global network information. A leap in the field of network embedding was taken when B. Perozzi et al. published their work, DeepWalk [8]. Their technique was heavily inspired by recent advancements in natural language processing where Mikolov et al. introduced the Skip-Gram model, word2vec [16], which had the ability to embed words (from sentences) in continuous vector spaces. Perozzi's main contribution was to consider vertices in a network similar to words in a sentence, and found that he could obtain accurate embeddings of networks.

Since then, many researchers followed and a generalization of DeepWalk was published under the name node2vec [17]. Both DeepWalk and node2vec were based on a random walk procedure in the network to which node2vec contributed by adding a tunable bias in the walks allowing to configure the trade-off between homophily and structural equivalences.

More prominent work in the field of network embedding is worth referencing, here among LINE [10], SDNE [11], and DNGR [12]. The latter 2 are very recent techniques relying on deep learning. Both are further described in Section 5.2.

Matrix factorization techniques are widely used for link prediction, especially in the domain of recommender systems where such techniques can extract latent features to perform preference predictions [14]. In a paper by D. Liang et al. [18] they extend a standard matrix factorization technique, introducing regularizing by entity co-occurrence counts, and thereby outperform another high-performing matrix factorization technique (Weighted Matrix Factorization [19]). They empirically demonstrate their model's superiority on 3 dataset similar to the datasets included in our experiments.

4. Problem Definition

In this section, we first describe a conceptual view of the challenges ahead. Then we formalize multiple subproblems that will be solved later, in Section 5. The remainder of this paper will extensively reference our formalization while going deeper into individual solutions.

4.1. Conceptual Definition

We approach the task of link prediction in (un)weighted bipartite networks. For our input, an undirected bipartite network, we require the weights of edges to be non-negative representations of the relation-strength between 2 vertices.

Without loss of generality we focus our efforts on networks common for social, economic and information systems. Such networks may contain information on preferences, opinions, and collaboration. Hence, an example of edge weights in our input network may be a discrete number of purchases (or views) of products by users in e-commerce. Another example may be a binary value representing whether a scientist has authored a specific papers – or not.

We consider the link prediction problem to be an equivalent of predicting relations between pairs of entities. E.g. given incomplete information, are we able to predict whether a scientist has authored a specific paper? Or if a user is likely to purchase a specific product? etc..

Our approach to link prediction contains network embedding methods. Given a network representation we must be able to compute low-dimensional vector representations of its vertices. Depending on the dataset, these vectors may be representations of users, products, scientists, papers, etc..

A third and final sub-problem is the link prediction where we seek utilization of feature vectors to rank vertex pairs proportionally to their likelihood of being connected in the future.

The problems described above will be formalized and further described in the following sections.

4.2. Data Representation by Networks

In Figure 1 we have a weighted network as input which we denote $G(V, E, W)$ (or simply G), having nodes $V = \{v_1, \ldots, v_n\}$ and edges $E \subseteq (V \times V) = \{(v_i, v_j)\}_{i,j=1}^{n}$. We may further extend the notation to capture the fact that it's bipartite: $G(V_A, V_B, E, W)$ where V is partitioned such that $V = V_A \cup V_B$ and $V_A \cap V_B = \emptyset$.

For each vertex pair in E there exist an element in the adjacency matrix A such that

$$a_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Furthermore, for every edge there exists a positive weight in W such that $w_{ij} > 0 \leftrightarrow a_{ij} = 1$, otherwise $w_{ij} = 0$

Our first task is to create a proper representation of the information contained in the network. The main requirement for the new representation is that it must be a network with non-negative weights – no requirements to whether it's directed or bipartite. One such network we denote by G'.

4.3. Network Embedding

We will consider the task of embedding networks, which is defined in the following way.

A function, f, maps vertices of an arbitrary network G' to a d-dimensional space, i.e.

$$f : v_i \mapsto \mathbf{x}_i \in \mathbb{R}^d \tag{2}$$

where $v_i \in V'$ and $d \ll |V'|$. Hence, $X \in \mathbb{R}^{|V'| \times d}$

4.4. Link Prediction

A third task is to predict links in G, given vectors that are representative of a pair of vertices.

Using network embedding methods, we approximate the weights W of the input network G such that we obtain approximation $\hat{G}$ having weights $\hat{W}$.

A function, g, maps the representative vectors of a pair of vertices to real number, i.e predicting the edge weight between $v_i \in V_A$ and $v_j \in V_B$.

$$g : (\mathbf{x}_i, \mathbf{x}_j) \mapsto \hat{w}_{ij} \in \mathbb{R} \tag{3}$$

Ranking of Vertex Pairs

In practice we are often interested in the relative ranking of vertex pairs rather than binary link predictions. Most methods for link prediction assigns a score to pairs of unconnected vertices proportional to the likelihood of existence of a link between them [14].

Therefore, the objective of g in Eq. 3 is to compute predicted edge weight, $\hat{w}_{ij}$, proportional to the likelihood of the (future) existence of edge (v_i, v_j).

5. Method

Figure 1 is a conceptual overview of our solution to subproblems defined in Section 5. It is an algorithmic framework for predicting links in a bipartite network, using

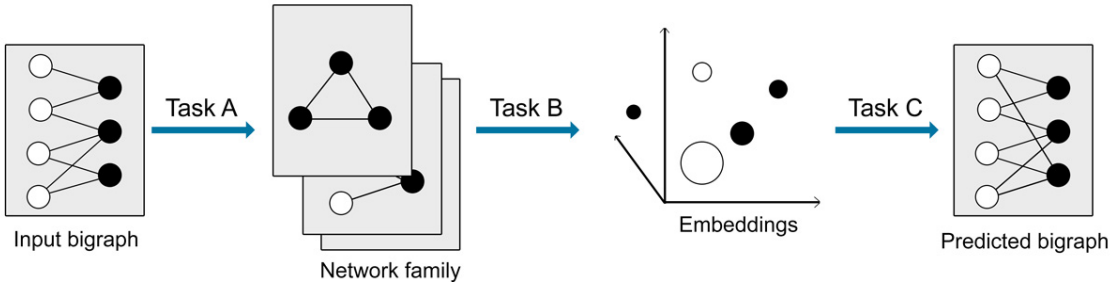

Figure 1: *Conceptual illustration of proposed framework: From raw data to link prediction.*

network embedding algorithms on a family of networks. The 3 arrows (denoted task A, B and C), represent specific tasks that we will solve in respectively Section 5.1, 5.2, and 5.3.

A key feature of our framework is its modularity. Replaceable modules allows for the framework to be updated continuously as research advances in related fields. For example, we allow for replacement of network embedding techniques.

Our solution to problems defined in Sec. 4 involves first constructing a family of networks, representing input data. Then, for every network, we compute network embeddings which we utilize to predict links.

In the following sections, we provide details on our methods together with important theory underlying the employed techniques. Section 5.1 explains the construction of derived networks from the input network. Section 5.2 describes the employment of network embedding methods. Section 5.3 further describes the utilization of network embeddings to predict links, and Section 5.4 analyzes the algorithmic complexities of methods described.

5.1. Network Topologies

As input to our framework we have a weighted bigraph. This bigraph may not in its original representation be ideal for some network embedding methods. Therefore, on the basis of the input network, G, we are motivated to construct a family of networks, $\mathcal{G}$, having different data representations of the same information.

In the family of networks, $\mathcal{G}$, we have 2 classes: bipartite and unipartite. The bipartite networks are constructed by modifying edge set and weights of G such that the resulting network becomes: $G(V_A, V_B, E, W) \mapsto G'(V_A, V_B, E', W')$. The unipartite networks only contain the vertices of V_B and are therefore a projection of a bipartite network in the form $G(V_A, V_B, E, W) \mapsto G'(V_B, E', W')$. Figure 2 illustrates the concept of network projection.

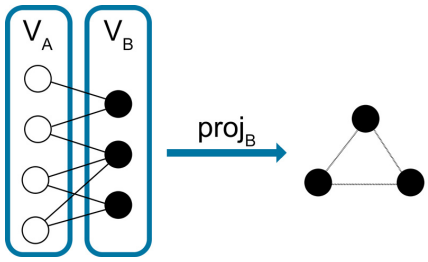

Figure 2: *The principle of projecting a bipartite network onto one partition of vertices, resulting in a unipartite network.*

The set of bipartite- and unipartite networks we denote $\mathcal{G}_{AB}$ and $\mathcal{G}_B$, respectively. We define the family of networks $\mathcal{G} = \mathcal{G}_{AB} \cup \mathcal{G}_B = \{G, G'_1, \ldots, G'_N\}$.

In total, we present 10 networks in in $\mathcal{G}$ that are either modifications or projections of G. All networks are listed in Table 1.

The 2 classes of networks will in the following sections be further described along with procedures for constructing individual network structures. Section 5.1.1 describes the motivation, properties and challenges in creating bipartite networks (#0-5) and Section 5.1.2 similarly describes for unipartite networks (#6-10).

5.1.1 Bipartite Network Variants

All bipartite networks in $\mathcal{G}$ contain all vertices in V_A and V_B and similarly all edges in E. The single difference is the weighting of edges, W.

In Table 1, network #0-5 are all bipartite.

Network #0 is the original weighted bipartite input and #1 and #2 are only variations hereof with scaled edge weights.

	#	Symbol	Pre-scaling	Post-processing	Weight calc.	Weight max.
Bipartite	0	G	-	-	-	∞
	1	G'_1	logarithmic	-	Eq. 4	∞
	2	G'_2	binary	-	Eq. 5	1
	3	G'_3	-	Normalization of weighted degrees	Eq. 6	1
	4	G'_4	logarithmic	*(same as above)*	Eq. 4, 6	1
	5	G'_5	binary	*(same as above)*	Eq. 5, 6	1
Unipartite	6	G'_6	-	Count of common neighbors	Eq. 7	∞
	7	G'_7	-	Jaccard index of adjacency vectors	Eq. 8, 9	1
	8	G'_8	-	Generalized Jaccard idx. of weight vectors	Eq. 10	1
	9	G'_9	-	Cosine similarity of weight vectors	Eq. 11	1
	10	G'_{10}	-	Projection by T. Zhou et al. [7]	Eq. 12	∞

Table 1: *Overview of all constructed networks in the family $\mathcal{G}$*

Network #1 has all edge weights scaled by

$$w' = \log(w + 1) \tag{4}$$

where $w' \in W'$ and $w \in W$. The addition of 1 is to avoid introduction of non-positive edge weights. This logarithmic scaling is motivated by the fact that many real-world networks are scale free, i.e. their degree distribution follows a power law. Hence, some vertices will have a degree several order of magnitudes higher than others vertices. Depending on the dataset and the employed network embedding algorithm we may wish reduce the power law's effect.

Network #2 has all edge weights converted to a binary value of 0 or 1. This is relevant for datasets where edge weights are of very little or no importance compared the knowledge of existence of edges, e.g. possibly, a user-product network should not attribute more importance to products having 3 views from a user compared to a single view – the important fact may lie in the difference between 0 and > 0 views. Binary conversion can be done easily by

$$w' = a \tag{5}$$

where a is the corresponding element in adjacency matrix, A.

Network #3-5 has the same pre-scaling as #0-2 in respective ordering but furthermore performs post processing on the scaled weights. This processing involves normalizing the weighted degrees for all $v \in V_A$ to 1. When this is performed the following holds

$$\sum_{v' \in \mathcal{N}_v} w(v, v') = 1 \tag{6}$$

where $\mathcal{N}_v$ is the set of neighbors to vertex $v \in V_A$ and $w(\cdot, \cdot)$ is the edge weight between two vertices.

This additional step of normalization is for reducing the influence of vertices with high weighted degrees and vice versa. For example, in a user-artist network of users listening to music artists we may not wish to have the most active music enthusiasts dominate our downstream prediction model completely. Users having only listened to 2-3 artists may also be of great value to our model. We may say that users are granted equal amounts of 'authority' in the network.

5.1.2 Unipartite Network Variants

Bipartite networks contain more information than their projections into unipartite networks [7]. However, a more compact network representation may help utilize certain network embedding algorithms.

We create a number of 5 unipartite network projections listed in Table 1 as network #6-10. These are all described below.

Network #6 is common in recommender system and is easily computed by setting edge weights between vertices $v_i, v_j \in V_B$ to the number of common neighbors they have in G (network #0)

$$w'_{ij} = |\mathcal{N}_i \cap \mathcal{N}_j| \tag{7}$$

where $\mathcal{N}_i$ is the neighbors of v_i and $\mathcal{N}_j$ the neighbors of v_j. w'_{ij} is the weight on the edge between vertices v_i and v_j in the projected network G'.

Network #7 uses the Jaccard index similarity measure to set edge weights. We specifically compute the similarity

between adjacency vectors $\mathbf{a}_i$ and $\mathbf{a}_j$ defined in Eq. 1. By the definition of Jaccard index we compute edge weights

$$w'_{ij} = J(\mathbf{a}_i, \mathbf{a}_j) = \frac{\mathbf{a}_i \cdot \mathbf{a}_j}{\|\mathbf{a}_i\|^2 + \|\mathbf{a}_j\|^2 - \mathbf{a}_i \cdot \mathbf{a}_j} \tag{8}$$

This measure of similarity ignores weights by computing on the column vectors of an adjacency matrix which are bit vectors.

An equivalent definition bears resemblance to Eq. 7

$$w'_{ij} = J(\mathcal{N}_i, \mathcal{N}_j) = \frac{|\mathcal{N}_i \cap \mathcal{N}_j|}{|\mathcal{N}_i \cup \mathcal{N}_j|} \tag{9}$$

Network #8 uses the generalized Jaccard index which includes information on weights contrary to the regular Jaccard index which works on binary data only, i.e. instead of working on adjacency vectors we consider weight vectors $\mathbf{w}_i$ and $\mathbf{w}_j$. Edge weights are computed by

$$w'_{ij} = \frac{\sum_{n=1}^{|\mathbf{w}_i|} min(w_{i,n}, w_{j,n})}{\|\mathbf{w}_i\|_1 + \|\mathbf{w}_j\|_1 - \sum_{n=1}^{|\mathbf{w}_i|} min(w_{i,n}, w_{j,n})} \tag{10}$$

where $w_{i,n}$ is the nth element in vector $\mathbf{w}_i$ (and similar for $\mathbf{w}_j$) and $\|\cdot\|_1$ is the L^1-norm of a vector.

Network #9 uses the cosine similarity measure which is commonly known for high performance in information retrieval, comparing text documents. As in network #8, we compute similarity between vectors $\mathbf{w}_i$ and $\mathbf{w}_j$. Cosine similarity defines weights by

$$w'_{ij} = cos(\mathbf{w}_i, \mathbf{w}_j) = \frac{\mathbf{w}_i \cdot \mathbf{w}_j}{\|\mathbf{w}_i\| \|\mathbf{w}_j\|} \tag{11}$$

where $\|\cdot\|$ is the L^2-distance of a vector.

Network #10 is built on a method by T. Zhou, et. al. [7]. The intuition behind their projection can be understood as a 'flow' of resources between vertices in network G. By assigning resources to a vertex, and running the resource-allocation process, the 'attractors' of resources are considered to be central for the starting vertex, which then becomes connected in G'. In their paper, weights are defined by

$$w'_{ij} = \frac{1}{k(v_j)} \sum_{v_n \in V_A} \frac{a_{in} a_{jn}}{k(v_n)} \tag{12}$$

where $k(\cdot)$ is the degree of a vertex.

Performance Considerations

Depending on the utilized dataset, some projections have a high risk of increasing the average vertex degree significantly which affects network complexity (in terms of $|E|$). This has consequences for both time- and space complexity. Our solution to this problem is limiting the vertex degrees to a fixed maximum, i.e. we only allow a fixed maximum number of edges per vertex selected by highest weight first. We empirically configured this upper degree limit to 25 such that no vertex in a unipartite network has more than 25 neighbors.

Unweighted Input Networks

Some datasets do not have weights on edges, which we interpret as a weight of 1 when a link exists, and 0 otherwise. In such cases, our network family $\mathcal{G}$ becomes smaller and only network #0 and #3 are kept in the group of bipartite networks. Furthermore, we can also omit network #8 as the general Jaccard index becomes 'regular' when $W = A$. In summary, for unweighted datasets $|\mathcal{G}| = 6$.

5.2. Network Embedding

Equation 2 defined f, mapping a vertex to a continuous vector of d-dimensions. In our framework, this method of dimensionality reduction is performed by one of 3 methods: node2vec [17], SDNE [11] and DNGR [12]. These methods have all demonstrated high accuracy in multi-class classification tasks on vertex labels, but has only superficially touched the link prediction problem. We intend to optimize the method's hyperparameters for link prediction tasks.

The 3 network embedding methods will be described in the following subsections.

node2vec (N2V) and DeepWalk

DeepWalk by B. Perozzi et al. [8] transfers the Skip-Gram NLP-model onto networks. The core of DeepWalk is the optimization problem:

$$\min_f -\log Pr(\{v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i-w}\} | f(v_i)) \tag{13}$$

where w is a constant, natural number, defining the *window* or *context*. The set of vertices (i.e. the context) is found by random walks in the network and then bounded by the window of size $2w + 1$. The equation seeks to learn a mapping f such that we maximize the probability of

predicting the context of v_i given it's embedding vector, $f(v_i)$. f is a matrix of size $|V| \times d$ for which we optimize parameters according to Eq. 13 using *stochastic gradient descent* (SGD).

node2vec by A. Grover et al. [9], is a generalized version of DeepWalk. They essentially propose an improvement involving flexibility in obeying 2 key principles: 1) representations must embed vertices from the same network community closely together, and 2) representations of vertices that share similar 'roles' in a network must have similar embedding – the former is useful in networks that are built on *homophily*, i.e. networks where similar entities has a tendency to bond. The latter is referred to as *structural equivalence* by the authors, where a pair of vertices, e.g. both being *hubs*, are considered to be structural equivalent.

node2vec sets to accomplish these principles by introducing a tunable bias in the random walk. Compared to DeepWalk, node2vec has 2 hyperparameters, p and q, biasing the walk towards capturing more (or less) of the homophily or structural equivalence properties. When both these parameters are unity, node2vec resembles DeepWalk.

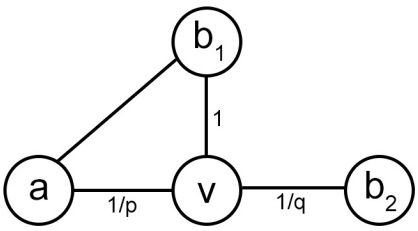

Figure 3: *node2vec random walk bias.*

The parameters influences the walk as illustrated in Figure 3: Consider that we have just traversed the edge (a, v) and we must decide what vertex to visit next, either b_1, b_2, or back to a. In the figure, numbers on the edges are factors denoted α, to which $w\alpha$ is the unnormalized transition probabilities (w being the edge weight). Let $d_{a,b}$ denote the shortest path distance between the previous and the next vertex, respectively a and b_i. Then, the formula for α is:

$$\alpha_{vb} = \begin{cases} \frac{1}{p}, & \text{if } d_{a,b} = 0 \\ 1, & \text{if } d_{a,b} = 1 \\ \frac{1}{q}, & \text{if } d_{a,b} = 2 \end{cases} \tag{14}$$

By tuning p and q, we may control how the walk progresses, e.g. biasing towards staying close to the starting node or exploring outwards.

Structural Deep Network Embedding (SDNE)

D. Wang et al. [11] suggest a deep approach to network embedding. Their intuition is to jointly optimize 1st and 2nd order network proximity by extending the loss-function of an autoencoder, whose input (and output) is vectors of the edge weight matrix, W, having length $|V|$. Such an autoencoder will be able to represent the neighborhood of a vertex (2nd order proximity). To include 1st order proximity, they extend the loss-function for learning autoencoder parameters with an objective to penalize distant embedding vectors for adjacent pairs of vertices, and similarly rewarding distant vectors for non-adjacent vertices.

Deep Neural Networks for Learning Graphs (DNGR)

S. Cao et al. [12] presents another deep approach bearing resemblance to both DeepWalk-inspired models and models such as SDNE. Cao's model replaces random walks with *random surfing* and restarts, heavily inspired by the PageRank algorithm [20]. They use the resulting probabilistic co-occurence matrix for creating vertex representations whose dimensionality is reduced by stacked denoising autoencoders.

5.3. Link Prediction

After having computed network embeddings stored in matrix X we then compute weight matrix $\hat{W}$ as introduced in Eq. 3 where $\hat{w}_{ij}$ was defined as the predicted weight on edge (v_i, v_j).

The procedure for computing $\hat{W}$ differs for embeddings of bipartite- and unipartite networks, respectively. The former has embedding vectors for both vertices in V_A and V_B. The latter is more difficult as we only have embedding vectors for vertices in V_B. Recapping Eq. 3; to find $\hat{w}_{ij}$ we require vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ where $v_i \in V_A$ and $v_j \in V_B$. For bipartite networks both of these embedding vectors can be found as columns in X. However, for unipartite networks we need to pre-compute $\mathbf{x}_i$ as this doesn't exist in X when vertices in V_A are not represented in the network. Section 5.3.1 and 5.3.2 elaborate on the complete procedure for link prediction. Section 5.3.3 extends the prediction problem to include multiple networks at once.

5.3.1 Bipartite Network Features

We define mapping g (Eq. 3) as the euclidean metric, i.e. $\hat{w}_{ij}$ is the L^2-distance between vector representations of a vertex pair where $v_i \in V_A$ and $v_j \in V_B$.

$$\hat{w}_{ij} = g(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| \tag{15}$$

5.3.2 Unipartite Network Features

In the beginning of Section 5.3 we brought attention to the issue of networks in $\mathcal{G}_B$ not containing vertices of V_A but only those in V_B. Therefore, in order to compute predicted weight as in Eq. 15 we pre-compute a representative vector of $v_i \in V_A$, still denoted $\mathbf{x}_i$.

We compute $\mathbf{x}_i$ as a weighted average of the embedding vectors in $\mathcal{N}_i$ which is the set of neighbor vertices of $v_i \in V_A$ in network G. The average is weighted by corresponding edge weights.

Consider an example with an e-commerce sales network of users and items. The networks in $\mathcal{G}_B$ would be networks of items-only projected from the user-item bipartite network G. In order to use the unipartite item-networks to predict user-item links we need to compute a user representation. Using the above method we would represent a user by the average of embedding vectors of items, observed to be linked with the respective user – and weighted by the strength of user-item links (e.g. number of purchases of a specific item).

5.3.3 Multiple Networks

In above sections it was described how weights are predicted by a single network in the network family, $\mathcal{G}$. However, predictive power of our framework is improved by using multiple networks. The procedure of using multiple networks for prediction is to construct feature vector, $\mathbf{y}_{ij}$, by concatenation of all single-network weight predictions, formally

$$\mathbf{y}_{ij} = (\hat{w}^0_{ij}, \ldots, \hat{w}^{|\mathcal{G}|}_{ij}) \tag{16}$$

where the superscript of $\hat{w}$ references the respective network in the family $\mathcal{G}$. Hence, we obtain a feature vector $\mathbf{y}_{ij} \mapsto \mathbb{R}^{|\mathcal{G}|}$.

The last step in the link prediction process is ranking predicted links. If we once again consider vertex pair (v_i, v_j) with feature vector $\mathbf{y}_{ij}$, we may then approximate $\hat{w}_{ij}$ with linear regression in the form:

$$\hat{w}_{ij} = \mathbf{y}_{ij}^\top \boldsymbol{\beta} + \mathcal{E}_{ij} \tag{17}$$

where $\boldsymbol{\beta}$ is a parameter vector containing the regression coefficients and $\mathcal{E}_{ij}$ is the error term.

As stated in Section 5.3, vertex pairs are sorted, ordered by likelihood of having (future) links between them.

5.4. Complexity Analysis

The scalability of our proposed framework rely entirely on its sub components. Let's consider the framework process from the beginning (recap Figure 1): Given a bipartite network as input we compute various modified and projected networks. The construction of these networks contributes with a linear factor ($|\mathcal{G}|$) to both space- and time complexities in all subsequent steps. However, our proposed network family has a fixed number of maximum 11 contained networks, allowing us to consider this complexity contribution as a constant.

The network construction phase, i.e. task A, involves constructing networks using various algorithms having different complexities. These networks were listed in Table 1. Network #1-5 can all be constructed with a time complexity of $\mathcal{O}(|E|)$ and a low constant factor. For all unipartite networks #6-10 we compute edge weights between every pair of vertices in V_B. Therefore, the time complexity is significantly higher, $\mathcal{O}(|V_B|^2 k_B)$ where k_B is the average degree of vertices in V_B. Though, the constant overhead for these computations are quite low with the last network, #10, being the most complex due to its resource-allocation process.

In the network embedding phase, task B, the employed methods all have time complexities linear to the number of vertices and avg. network degree, $\mathcal{O}(|V'|k)$ where k is the average degree of vertices in V', given that we consider dimensionality d fixed.

We may consider k (and k_B) as constants as many real-world datasets have a natural limitation in the avg. degree, e.g. in social networks the number of connected 'friends' are usually bounded. Furthermore, to further justify our assumption, during the network construction phase we set a fixed upper limit on our network degree with the purpose of avoiding unnecessary complexity (see section 5.1.2).

In the last step with linear regression, without subsampling training instances, we reach a time complexity of $\mathcal{O}(|E|^2)$, if we still consider the number of networks in $\mathcal{G}$ fixed. The constant factor in this last step is insignificant but we may choose consider less training samples while computing regression coefficients.

Overall, projecting graphs looks to have the highest time complexity without obvious countermeasures, and thereby being the bottleneck of our framework. Though, we observe the following: 1) The constant factor overhead is tiny to which employment on relatively large datasets are still feasible, 2) each of the projection methods are straight-forward to parallelize as they can all be locally computed.

6. Experiments

In this section, we evaluate proposed framework on multiple real-world datasets. Furthermore, we evaluate the performance of both a set of network structures and a set of network embedding algorithms to select the best performing configurations of our framework. We document the significant improvements over various baselines using a range of appropriate evaluation metrics.

6.1. Methodology

All optimization procedures and intermediate evaluation steps are computed using k-fold cross validation. We set $k = 5$ which we found to be a good trade-off between the execution time and the benefits with regards to avoiding overfitting and obtaining precise results. A '6th fold' was reserved for final tests, i.e. we partitioned all vertex pairs from datasets into 6 equally sized parts to which ∼83% was used for training and validation, and the remaining test data (∼17%) was used for a final evaluation.

All methods and experiments were performed in a Python environment on Linux, making use of trending libraries: Tensorflow, Keras, NetworkX, scikit-learn, etc. together with various author's implementations of their respective network embedding algorithms. Experiments were run using an Intel Core i7 6700K, 32 GB DDR4 RAM and a GPU, Nvidia GTX1080 which were utilized by 2 of 3 network embedding algorithms (SDNE and DNGR).

6.1.1 Datasets

A set of 3 network datasets are tested in our experiments. These datasets all contain bipartite network data and were selected to maximize coverage of the challenges within the research field of link prediction. The 3 datasets are:

- LastFM [21]: Contains music artist listening counts on Last.fm from a set of 2K users. This dataset has an avg. degree per user node of 49.
- Newman [22]: A network on arXiv co-authorship in the physics section on condensed matter. Binary links between authors and papers indicates authorship.
- Movielens [23]: A dataset containing 1M user-movie ratings on a 5-star scale. In line with [18], we convert this explicit rating to binary such that 4 and 5 stars becomes 1, otherwise 0.

The selected datasets covers various types of information we introduced in the beginning of this paper: Implicit preferences, explicit opinions, and collaboration. Statistics on the datasets are summarized in Table 2. *Rel.* indicates the underlying relationship type between vertex pairs.

	LastFM	Newman	Movielens
$\|V\|$	20K	39K	10K
$\|E\|$	93K	59K	1M
$\|V_A\|$	2K (user)	17K (author)	6K (user)
$\|V_B\|$	18K (artist)	22K (paper)	4K (movie)
Rel.	Listen count	Authoring	Explicit rating

Table 2: *Statistics on datasets.*

Pre-processing on Datasets

We mentioned our use of 5-fold cross validation and an extra holdout set for final testing. To properly validate and evaluate our model we need the users (or authors) from the datasets to always appear in both our training and validation set (or test set). Therefore, in our data we keep only users (or authors) having a minimum of 6 connected entities e.g. movies. To make sure each user are represented in every part including the test set, we systematically split user data uniformly into each partition.

Furthermore, we remove entities (artists, papers and movies) that have a very low degree. For our datasets we respectively remove artists listened to by fewer than 5 users, papers authored by only 1 scientist and movies rated by fewer than 5 users.

6.1.2 Baseline Algorithms

We consider the following 3 baseline models:

- *Random*: A model where links are predicted by a completely uniform randomization. The performance of this model is expected low and sets a lower bar for a better comparison perspective.

- *Popular*: A model where links are predicted entirely on the basis of the highest weighted degree for vertices in V_B. When networks are scale-free this model is expected to perform quite well.
- *Cofactor* [18]: A state-of-the-art matrix factorization model. This model extends regular matrix factorization of user-item matrices with regularization by co-occurrence matrices.

We consider the random- and popular model to be each others extremes in the way that the entropies of the respective lists of predicted results cannot be further apart. Though, the cofactor model is expected to be performing much better in precision.

6.1.3 Evaluation Measures

For evaluation of both intermediate steps in the optimization process and the final predicted ranking we compute the *normalized discounted cumulative gain* (nDCG), *mean average precision* (MAP) and *mean reciprocal rank* (MRR).

All of the 3 measures stays within the interval $[0, 1]$.

Normalized Discounted Cumulative Gain

nDCG is calculated by first computing the discounted cumulative gain:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{18}$$

where k defines a specific rank position and i therefore iterated over all the rank positions up to k. rel_i denotes the *relevance* of the instance at the ith position. The relevance can be binary: 1 for relevant documents and 0 for non-relevant documents – or it can be weighted by the edge weights in G. We consider the latter case.

The DCG is then normalized to nDCG by dividing by the ideal DCG (IDCG), i.e. the DCG scoring in the optimal setting with a perfect ranking.

$$nDCG_k = \frac{DCG_k}{IDCG_k} \tag{19}$$

Lastly, we compute the mean over all N queries.

$$mean(nDCG_k) = \sum_{i=1}^{N} \frac{nDCG_{k,i}}{N} \tag{20}$$

which we simply refer to as nDCG for brevity.

Mean Average Precision

MAP serves the same purpose as nDCG of evaluating ranked queries, however, the effect of discounting errors in lower rankings differs. As opposed to nDCG, MAP only considers binary relevance.

First, we compute the average precision over the top k ranking positions

$$AP_k = \sum_{i=1}^{k} \frac{P(i)}{min(m, k)} \tag{21}$$

where $P(i)$ denotes the precision of the top i ranking positions and m is the total number of relevant items to be retrieved.

Then, we calculate the mean of average precisions over all N queries by:

$$MAP_k = \sum_{i=1}^{N} \frac{AP_{k,i}}{N} \tag{22}$$

Mean Reciprocal Rank

A final member belonging to the class of ranked query evaluation is the MRR. As opposed to nDCG and MAP, MRR only considers the first relevant document in a query

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i} \tag{23}$$

where $rank_i$ is the rank of the first relevant document. If there are no relevant documents then reciprocal rank is set to zero.

Measure Properties

NDCG, MAP and MRR are all measures to evaluate on rankings and they share the common property of rewarding relevant items in the top of the lists. MRR only considers the top ranked item and is useful when we are only interested in the best result of a query, whereas NDCG and MAP considers more items with a decay on the rank position of items.

In this paper we mostly prefer NDCG (over MAP) due to its ability of distinguishing between real valued relevance rather than only binary relevance. Though, we include MAP for 2 reasons: 1) because of it's strong establishment in related research fields, and 2) in some cases where relevance scores are not compatible, NDCG can be misleading, and binary relevance is more fit.

We set $k = 100$ for all measures in our experiments to limit the computational cost of evaluation. Any increment of k would have insignificant influence on the prediction score due to discounting in each of the measures. For brevity we denote NDCG_{100} simply as NDCG, and similarly for MAP and MRR.

6.1.4 Network Embedding Optimization

Our framework contains multiple learning models which needs optimization. In this Section, we describe the search for optimal hyperparameters influencing how mapping f is learned.

Each of the network embedding methods have hyperparameters to be optimized, including vector dimension d. The objective for our employed network embedding methods are to accurately predict relations on vertex pairs where an edge is not (yet) observed. To evaluate such predictions we apply k-fold cross validation such that network embeddings are computed for $k-1$ folds and evaluated on the validation fold. For bipartite graphs, this is equivalent to removing $\frac{1}{k}$ of edges during training and afterward evaluating on this holdout set of edges. For unipartite networks this is more abstract as we cannot simply remove edges in the training set. However, we still construct a network using the training data and then construct an entirely new network of the validation data for evaluation.

We never evaluate the embedding methods on the training data as the methods are very prone to overfitting, especially when d is high compared to the dataset. Therefore, we select the hyperparameters that perform best averaged over k-folds. As an alternative to grid- or random search algorithms we use *tree of parzen estimators* (TPE) [24] for finding the best parameters. We allow for completely different parameters for each network in $\mathcal{G}$, including d.

Regression Analysis

In this Section, we describe the procedure for performing variable (feature) selection. Eq. 16 defined the feature vector $\mathbf{y}$. This feature vector is fitted by linear regression in Eq. 17. Linear regression has proven to be efficient for the output of network embedding techniques, but in our case may suffer from a collinearity problem where multiple of the variables are highly correlated. Furthermore, unregularized linear regression coefficients suffers from numerical instability and becomes sensitive to noise.

We employ the regression analysis method, *least absolute shrinkage and selection operator* (lasso) [25], to regularize the regression coefficients. Lasso (L_1-regularization) has the advantage over L_2-regularization in its ability to set (some) coefficients to exactly 0, and thereby making it possible to perform variable selection simply by scaling the regularization term.

Class Balancing

For our model to be able to predict existence of links we must also learn it when links are not existing. Therefore we supplement the edge data used for learning regression coefficients with an equal amount of randomly sampled non-edges (unconnected vertex pairs).

We experimented with sampling k-times more non-edges and weighting those samples by the reciprocal of k, however, the small gain in model stability was not worth the increased computation time on tested datasets.

6.2. Experiment Results

In this Section, we will present the results of proposed framework contra baseline models. We will begin by documenting the variable selection. Next, we document best performances of fully tuned models, and later dig deeper into the parameter sensitivity, visualizations, etc..

6.2.1 Variable Selection

Another name for this section could be 'Network Selection' due to the fact that filtering variables, i.e. setting regression coefficients to zero, allows us to omit the entire corresponding network from being constructed. When we e.g. keep only a small subset of variables for prediction we significantly reduce complexity and thereby computational load.

There exists a special case in selecting only a single variable, which would not only reduce the effort in our network creation phase to a minimum, but also eliminate the need for linear regression due to the single variable in itself being sufficient for ranking – a constant factor (regression coefficient) can at most reverse the ranking.

In Figure 4, the performance (NDCG) is plotted as a function of the number of selected variables (or networks). The results in the scatter plot is from the LastFM dataset using node2vec as the network embedding method. The variable selection experiment was conducted by adjusting the lasso regularization term. Observe that performance is

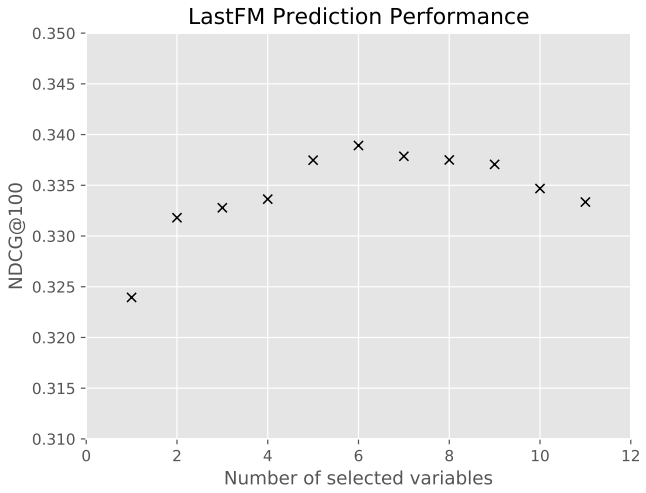

Figure 4: *Performance of link prediction on the* LastFM *dataset using node2vec. Number of selected variables are varied by adjusting the lasso regularization term.*

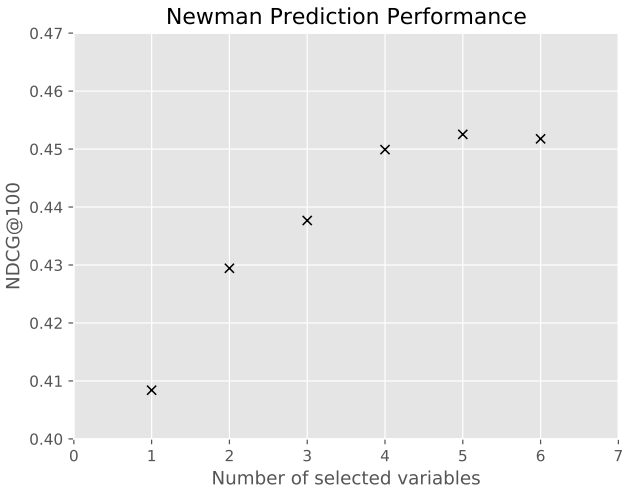

Figure 5: *Performance of link prediction on the* Newman *dataset using node2vec. Number of selected variables are varied by adjusting the lasso regularization term.*

best when only 6 networks are utilized. Further inclusion of networks leads to more noise, affecting performance.

In Figure 5, we perform the same experiment to select the optimal number of variables, but for the Newman dataset. Here, there are fewer variables to choose among because the dataset contains binary edge weights and as mentioned in Section 5.1.2, many of the networks becomes redundant. Observe that 5 utilized networks is the optimal number for this specific dataset.

In the Movielens dataset, the 4 variables selected were almost similar to those for the LastFM dataset, employing both unipartite- and bipartite networks.

6.2.2 Link Prediction

Performances are summarized for each model in Table 3 (below). Our proposed framework has been abbreviated LPFB (Link Prediction Framework for Bigraphs). The subscript in LPFB_1 indicates we are employing only the best performing network in $\mathcal{G}$ – this was regarded as a special case in Section 6.2.1. Subscript: $*$ indicates an optimal configuration of our framework, after performing variable selection.

Dataset	Method	MAP	NDCG	Rec.	MRR
LastFM	Random	0.002	0.011	0.039	0.011
	Popular	0.040	0.140	0.277	0.145
	Cofactor	0.125	0.302	0.504	0.366
	LPFB_1	0.139	0.326	0.588	0.381
	LPFB_*	**0.147**	**0.339**	**0.607**	**0.393**
Newman	Random	$\sim$0	0.002	0.010	$\sim$0
	Popular	0.001	0.004	0.015	0.003
	Cofactor	0.195	0.319	0.677	0.249
	LPFB_1	0.292	0.408	0.682	0.368
	LPFB_*	**0.373**	**0.479**	**0.707**	**0.457**
Movielens	Random	0.002	0.016	0.033	0.022
	Popular	0.058	0.195	0.336	0.258
	Cofactor	**0.126**	**0.335**	**0.551**	**0.396**
	LPFB_1	0.097	0.275	0.461	0.333
	LPFB_*	0.117	0.319	0.513	0.364

Table 3: *Performance of proposed framework compared to baselines.*

Observe the exceptional results of our model for the datasets LastFM and Newman, for which we improve by respectively 18% and 96% over the best baseline model. Even the simplistic special case, LPFB_1, demonstrates superiority in predictive abilities. Compared to this special case model, we are able to perform variable (feature) selection on our network family and improve performance

on the 3 datasets by respectively 6%, 28%, and 21%, which demonstrates the usefulness of constructing multiple network structures.

For the MovieLens dataset, the Cofactor baseline model is 8% better performing (in terms of NDCG). The potential cause of our framework not performing well on this dataset might be due to improper modeling of the bipartite input network. We have recreated the same preprocessing of ratings as suggested by the authors of the Cofactor model, in which only rating of 4 stars and higher are kept as a binary indication that a user likes a movie – this might be optimal for their algorithm, however, we believe that the great loss of information is not optimal for our framework, which has a strength in learning by itself, what's important and what's not.

We expect the *random* model's performance to be inversely proportional to the size of V_B. This is the reason for the decrease in performance in Newman compared to LastFM and MovieLens.

The *popular* model performs relatively well on datasets containing preferences and opinions (LastFM and MovieLens). However, it performs poorly on the Newman collaboration dataset due to the nature of how this network was formed. In contrast to the other datasets, this specific dataset is not expected to have formed under the influence of preferential attachment rules ('the rich gets richer'), which underlies the formation of scale-free networks. In other words, a paper does not have a tendency to be authored by more scientists simply due to the current number of authors the same way as popular music tends to attract more listeners.

In Figures 6 to 8, the precision at top-k positions is plotted with a varying k. It is notable how the Newman results are very different. This is due to the dataset not being formed according to popularity and preferential attachments, where the *popularity* model has almost no predictive power. Apparently, the highly complex nature of our framework yields superior results in such cases, whereas the Cofactor model is more 'popularity'-driven.

6.2.3 Network Embedding Results

In Table 4 we evaluate the performance of individual network embedding methods.

Emb. algo.	Dataset	MAP	NDCG	Rec.	MRR
N2V	LastFM	**0.147**	**0.339**	**0.607**	**0.393**
	Newman	**0.373**	**0.479**	**0.707**	**0.457**
	MovieLens	**0.117**	**0.319**	**0.513**	**0.364**
SDNE	LastFM	0.078	0.222	0.444	0.208
	Newman	0.292	0.385	0.598	0.360
	MovieLens	-	-	-	-
DNGR	LastFM	0.088	0.244	0.518	0.253
	Newman	0.097	0.173	0.397	0.125
	MovieLens	-	-	-	-

Table 4: *Best performances of different network embedding algorithms in proposed framework.*

node2vec performs best on all datasets to which we attribute both its flexibility in adapting to network's topological properties (by tuning the random walk bias), but also to its relatively more successful hyperparameter tuning.

Due to some difficulties with the methods SDNE and DNGR, explained in the following, we have omitted the MovieLens results to focus on more accurate results in the remaining 2 datasets.

SDNE and DNGR are deep approaches having multiple hidden layers in their autoencoders. The task of learning parameters of such models is computational expensive. Our setup relied on a GPU for higher efficiency, however, we met issues in the memory management of GPU's, probably due to early development (beta) phases of required drivers – thus, running experiments on our GPU-setup required almost constant supervision.

6.2.4 Network Embedding Visualization

We can visualize embeddings in 2D by further reducing dimensionality on our embedding space using the *t-SNE* algorithm [26].

In Figure 9, embedding vectors of network #0 from the LastFM dataset are plotted. Colored dots are representing artists, and black crosses represent users. Both users and artists are distributed in the embedding space, indicating that every artist has an audience and vice versa. Density-clustered points are visually apparent, most likely due to the formation of sub-communities having a similar taste in music.

The red, yellow, and blue coloring of the artist points are assigned user-tagged labels, respectively for music gen-

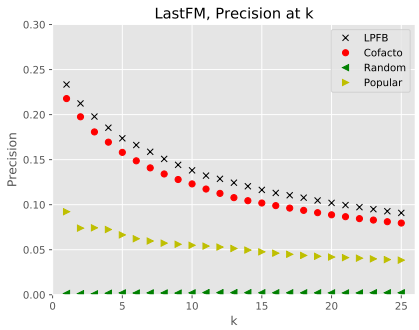

Figure 6: LastFM *prediction precision at top-k positions.*

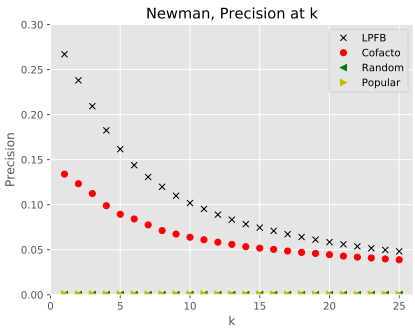

Figure 7: Newman *prediction precision at top-k positions.*

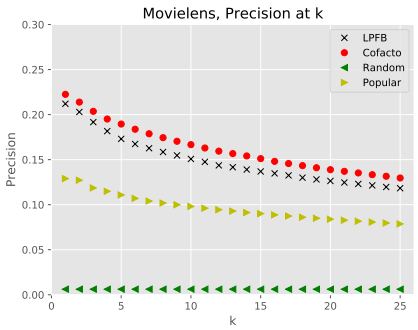

Figure 8: Movielens *prediction precision at top-k positions.*

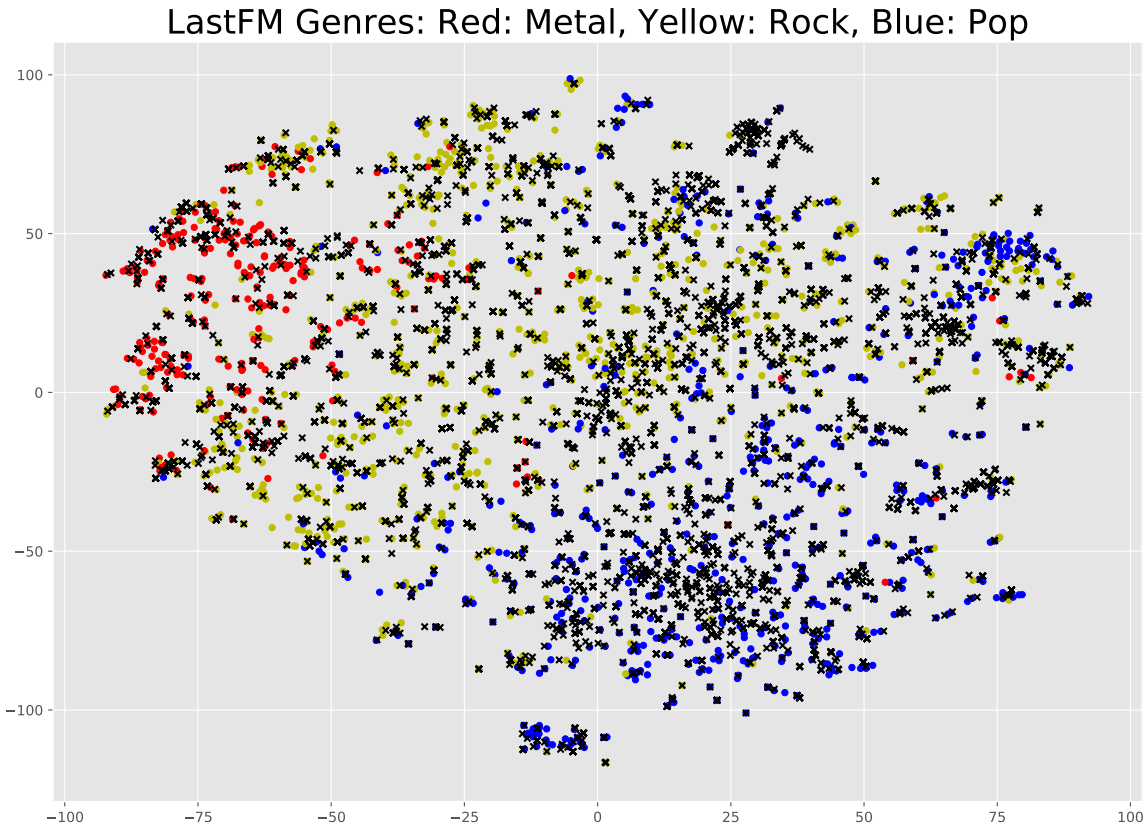

Figure 9: *t-SNE visualization of embedding space of network #0 from* LastFM *dataset. Points (embedding vectors) have been labeled with user-assigned tags according to music genres. Users and artists are represented by respectively black crosses and colored dots.*

res metal, rock, and pop. Notice the separation between metal and pop artists with rock artists as the separator, which presumably is well-representative of real-world relations between genres, e.g. pop-rock or metal-rock pairing are probably more common than pop-metal – Not to be confused with the actual genre of 'pop metal', which has very little to do with pop.

Besides genres, there are more latent relations between artists and users we are able to represent by our unsupervised feature learning approach. Conveniently, the embedding vectors are a bi-product of our framework to be employed in other downstream machine learning tasks other than link prediction, such as multilabel classification of genres.

6.2.5 Parameter Settings

Table 5 is an example of our bias-parameter settings for the entire list of embedded networks for the LastFM dataset. The parameters were introduced for node2vec in Section 5.2. Other parameters were also tuned, including number of, length, and window size of random walks, however, these did not deviate significantly from the proposed settings by the node2vec authors.

Table 5: *Parameter settings of walk bias in node2vec*

Network #	In-Out Param. (q)	Return Param. (p)
0	0.09	5.36
1	0.77	0.40
2	5.75	4.16
3	0.15	9.07
4	0.96	1.29
5	1.25	4.87
6	0.19	0.08
7	0.37	0.27
8	0.20	12.7
9	0.07	0.29
10	19.8	0.05

Some notable observations can be made from Table 5: 1) Networks with binary edges have a tendency to have a high setting of q, thereby making random walks more inclined to staying near the starting point, and 2) both parameters for network #10 adjusted to extremes which is notable due to this network being the only directed, which we may conclude affects behavior significantly.

7. Further Work

Future work may involve finding techniques for reducing connectivity of constructed networks as a blowup in network density causes a proportional rise in computation time. We empirically set an upper bound of vertex degrees on 25, however, we would like to achieve greater insight into the effect of this restriction with respect to final prediction performance.

Furthermore, in the network construction phase, we observe that many networks are constructed by weight-equations being much similar, to which we could look into constructing a generalized equation with tunable parameters to construct network edge weights. Besides potentially improving prediction this further work could make the network construction phase more appealing and elegant.

Overall, of the 3 network embedding methods, node2vec performed best. However, SDNE and DNGR might not be performing to their full potential for multiple reasons: 1) The SDNE-implementation was sent directly by the authors having issues with non-working code fragments and solutions inconsistent with their published paper. Issues were corrected but at the risk of not recreating the high performance model the authors intended, and 2) SDNE and DNGR were both able to utilize the performance of GPU's but as we had come to realize, such utilization of GPU's suffers from instability and crashed constantly due to poor GPU-memory handling. This made it difficult to tune hyperparameters which may therefore not be optimal.

We demonstrated the usefulness of the lasso regression technique for feature selection. However, lasso inconsistently selected different variables when data was given as input in random order. Future work should involve improving variable selection either by running lasso multiple times to statistically determine the best variables – or trying other approaches.

8. Discussion and Conclusion

In this paper, we studied the employment of network embedding methods with the purpose of performing link prediction in bipartite networks. We took a general approach relying only on topological data, thereby making our proposed framework applicable to various fields regardless of what is being modeled.

We built a family of networks with a purpose of generating different representations of information contained in

a bipartite network on which we wish to predict links. We carefully motivated and designed ten networks derived from the bipartite input network. Then, we optimized a mapping to compute optimal vector representations for each vertex in each network belonging to our family. By means of regression analysis, we selected a subset of networks that had maximal predictive power for link prediction, to which we formed a ranking of vertex pairs being proportional to the likelihood of having an existing edge between them.

We performed experiments on three datasets consisting of different types of modeled information: (implicit) preferences, (explicit) opinions, and collaboration. We demonstrated our framework's efficiency over various baseline models, including state-of-the-art matrix factorization. Furthermore, by means of visualization, we argued for the empiric usefulness of computed embeddings for other unrelated machine learning tasks such as multilabel classification.

In our experiments, we compared three recent network embedding methods to which node2vec performed best. Though, tuning of hyperparameters for the other models, SDNE and DNGR, could be improved in future work to provide a more 'fair' performance comparison.

In the LastFM and Newman datasets we demonstrated significant improvements in predictive abilities on respectively 18% and 96% (measured in NDCG-scores). For the Movielens dataset, however, the Cofactor baseline method were 8% better. For this latter dataset, we argued that our framework might see improvements by preprocessing input network information in a way more suitable for our methods.

References

[1] R. Diestel, *Graph theory {graduate texts in mathematics; 173}*. Springer-Verlag Berlin and Heidelberg GmbH & amp, 2016.

[2] X. Li and H. Chen, "Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach," *Decision Support Systems*, vol. 54, no. 2, pp. 880–890, 2013.

[3] J. Schwender and B. H. Junker, *Plant metabolic networks*. Springer, 2009.

[4] K.-I. Goh and I.-G. Choi, "Exploring the human diseasome: the human disease network," *Briefings in functional genomics*, vol. 11, no. 6, pp. 533–542, 2012.

[5] Y.-Y. Ahn, S. E. Ahnert, J. P. Bagrow, and A.-L. Barabási, "Flavor network and the principles of food pairing," *Scientific reports*, vol. 1, 2011.

[6] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*. Springer, 2011.

[7] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang, "Bipartite network projection and personal recommendation," *Physical Review E*, vol. 76, no. 4, p. 046115, 2007.

[8] 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*, pp. 701–710, ACM, 2014.

[9] 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*, pp. 855–864, ACM, 2016.

[10] 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*, pp. 1067–1077, ACM, 2015.

[11] 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*, pp. 1225–1234, ACM, 2016.

[12] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1145–1152, AAAI Press, 2016.

[13] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[14] V. Martínez, F. Berzal, and J.-C. Cubero, "A survey of link prediction in complex networks," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 69, 2016.

[15] M. Al Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*, pp. 243–275, Springer, 2011.

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

[17] 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*, pp. 855–864, ACM, 2016.

[18] D. Liang, J. Altosaar, L. Charlin, and D. M. Blei, "Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence," in *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 59–66, ACM, 2016.

[19] K.-Y. Chen, H.-M. Wang, B. Chen, and H.-H. Chen, "Weighted matrix factorization for spoken document retrieval," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8530–8534, IEEE, 2013.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," tech. rep., Stanford InfoLab, 1999.

[21] I. Cantador, P. Brusilovsky, and T. Kuflik, "2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011)," in *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, (New York, NY, USA), ACM, 2011.

[22] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, 2001.

[23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2016.

[24] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

[25] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[26] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.