A Survey of Link Prediction Techniques

Vivian Feng

Thomas Jefferson High School for Science and Technology

Computer Systems Research Lab

Mr. Kosek

December 2, 2022

Abstract

The problem of link prediction, predicting if two nodes in a network have a connection between them, is a theoretical problem with many field-agnostic real-world applications. This paper investigates the efficacy of three sets of link prediction algorithms: local node similarity heuristics, the global Random Walk with Restart index, and Node2Vec embedding. This proposal aims to provide insight into the performance of canonical link prediction algorithms on small graphs. The graphs are sampled from various domains and include infrastructure and ecological networks.

A Survey of Link Prediction Techniques

Purpose

Graphs can represent road networks, airplane flights, and ecological and social relationships. The task of link prediction, determining whether two unconnected nodes in a graph have an edge between them, is a theoretical problem with field-agnostic real-world applications. For example, Facebook uses link prediction to suggest people users may know (Facebook, n.d.). Node embeddings can be used to identify potential lateral attacks in computer networks (Bowman, 2022). Because the structures of real-world networks vary greatly, no single link prediction algorithm is a panacea. Different link prediction algorithms have different efficacies. This study will investigate the efficacy of local similarity indices, random walk with restart global index, and Node2Vec on graphs not traditionally used in academic research. Most published work tests the performance of link prediction algorithms on large graphs with thousands of nodes and millions of edges. This project will provide more information on the performance of link prediction algorithms on small graphs.

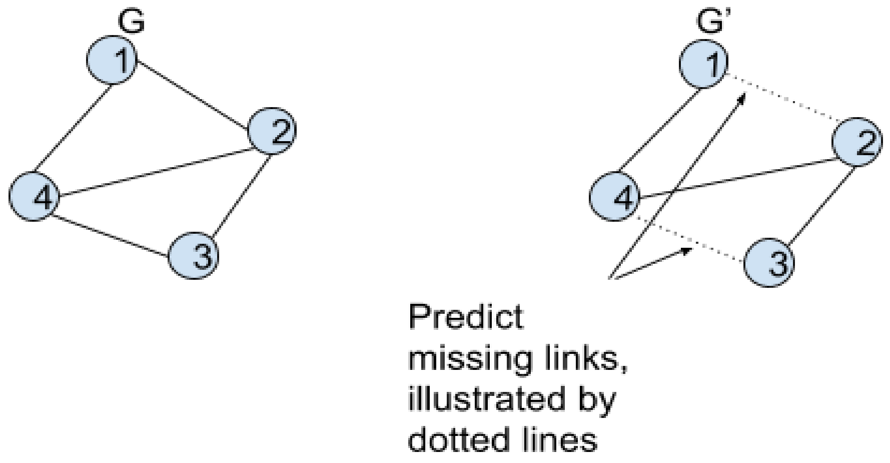
Background

I. Problem Definition

A graph is a collection of nodes linked together by edges. In this proposal, the terms “vertex” and “node” will be used interchangeably. Likewise, the term “graph” and “network” will also be used interchangeably.

The link prediction problem aims to find unobserved edges in an incomplete version, *G’*, of the entire graph *G*. Refer to Figure 1 for a graphical representation of the problem.

Figure 1. Graphical representation of link prediction problem. Original work of the author.



I.1 Evaluation metrics

This proposal will use the Area Under the receiver operating Curve, AUC, as the measure of accuracy. Algorithmically, this proposal will estimate the AUC value using the formula (Lü & Zhou, 2011, p. 1152):

$$AUC = \frac{n' + 0.5n''}{n}$$

AUC in the case of link prediction gives the probability that an existing link has a higher score for an index value than a nonexistent link. Out of *n* independent random selections of edges, *n'* is the number of times that an existing edge has a higher score than a nonexistent edge, and *n"* is the number of times that an existing edge has a score less than or equal to that of an absent edge.

II. Local Similarity Indices

Proximity-based indices predict the existence of a link based on how many shared neighbors they have. Certain similarity index formulations apply a weight based on degree. The following sections detail some widely used similarity indices.

II.1 Common Neighbors (CN)

Common Neighbors (CN) Index is denoted by s_{cn}. Suppose there are two vertices *x* and *y*. If N(x) denotes the set of node *x*'s neighbors, |Q| denotes the number of elements contained in set Q, and ∩ denotes the intersection of two sets, then the value of the Common Neighbors Index is (Lü & Zhou, 2011, p. 1153):

$$s_{cn} = |N(x) \cap N(y)|$$

II.2 Hub-Promoted Index

The Hub-Promoted Index (s_{HP}), where k_x is the degree (number of neighbors of a node) of node x, is defined as (Lü & Zhou, 2011, p. 1154):

$$s_{HP} = \frac{|N(x) \cap N(y)|}{min(k_x, k_y)}$$

By dividing by the minimum degree, links that are close to hubs tend to be assigned higher scores.

II. 3 Hub-depressed Index

The Hub-Depressed (HD) index is defined as (Lü & Zhou, 2011, p.1154):

$$s_{HD} = \frac{|N(x) \cap N(y)|}{max(k_x, k_y)} \cdot$$

Links further from hubs have higher scores.

II.4 Leicht-Holme-Newman index

The Leicht-Holme-Newman Index is defined as (Lü & Zhou, 2011, p.1154):

$$s_{LHN} = \frac{|N(x) \cap N(y)|}{k_x \times k_y}$$

The denominator is thought to be proportional to the expected number of nodes and act as a scaling factor on the index.

II.5 Adamic-Adar index

The Adamic-Adar Index is defined as (Lü & Zhou, 2011, p.1154):

$$s_{AA} = \sum_{i \in N(x) \cap N(y)} \frac{1}{log(k_i)}.$$

More weight is placed on less-connected neighbors. The Adamic-Adar index was initially proposed for use in social networks.

II.6 LHN variation

This proposal defines a novel variation of the LHN index: s = $\frac{|N(x) \cap N(y)|}{log(k_x \times k_y)}$.

The log factor attempts to smooth out differences in degree.

III. Random Walk with Restart

Global similarity indices, such as the Random Walk with Restart (RWR) Index, assign a score to a candidate link by factoring in all the nodes' properties. The RWR index is derived from the steady-state probability of a random surfer that is wandering from node to node, with probability *c* that they will go to a neighboring node and probability *1-c* that they return to the starting node. It is assumed that the surfer will choose any neighbor with equal probability. The transition matrix is defined as P, where P_{ij} is $1/k_i$ if i and j have an edge, else $P_{ij} = 0$ (Lü & Zhou, 2011, p.1156). P_{ij} refers to the element in matrix P at row *i*, column *j*.

At steady state, the probability of where the surfer will end up starting at node x does not change. If $\mathbf{q}_x$ is this probability vector, where entry *y* denotes the probability that the surfer will eventually end up at node *y* starting at *x*, then (Lü & Zhou, 2011, p.1156):

$\mathbf{q}_x = cP^T\mathbf{q}_x + (1 - c)\,\mathbf{e}_x$ ($\mathbf{e}_x$ denotes the x-th column of the identity matrix which has the same dimensions as P)

Solving for $\mathbf{q}_x$ finds that $\mathbf{q}_x = (1 - c)(I - cP^T)^{-1}\mathbf{e}_x$.

The index is then defined as $q_{xy} + q_{yx}$, where q_{xy} is *y*-th element of vector $\boldsymbol{q_x}$ (Lü & Zhou, 2011, p.1156).

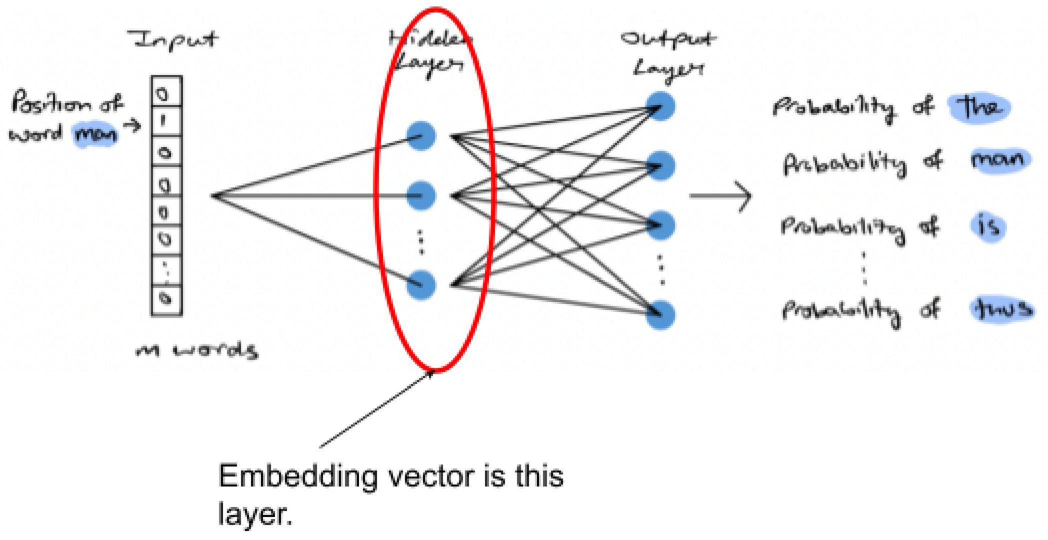
IV. Node2Vec

In recent years, advancements in machine learning have led to better low-dimension representations of graphs in vector space. These vector representations are known as embeddings. Grover and Leskovec (2016) applied the Skip-gram model, initially developed for generating word embeddings in natural language processing applications, to graphs.

IV. 1 Explanation of Node2Vec Architecture

The Skip-gram model generates an embedding of a corpus of words in lower-dimensional space by trying to predict what words fall within a chosen word's context window. The embedding feature vector is the hidden layer of the one-layer neural network used to generate the probability vector for words in the corpus that will fall in the given word's context window (Vatsal, 2022). Figure 2 illustrates the location of embeddings.

Figure 2. Location of embeddings. Image modified from Vatsal (2022).



Analogously, the Skip-gram model can be applied to graphs. In the algorithmic framework Node2Vec, Grover and Leskovec (2016) developed a mapping of nodes to feature vectors in embedding space by applying the Skip-gram model to sequences of nodes sampled by weighted random walks of fixed lengths. Node2Vec treats random walks as "sentences" and nodes as "words" when inputting them into the Skip-gram model.

Using the random walks, the model learns the embedding vectors via stochastic gradient descent. Compared to other methods of node sequence sampling, such as DFS and BFS, random walks have a lower big-O space and time complexity (Grover & Leskovec, 2016).

IV.2 Visualization Node2Vec embeddings

Combined with dimensionality reduction algorithms such as Principal Component Analysis, the embeddings representing each node in a graph can be used to create visualizations that allow for easy identification of clusters of related nodes (Vatsal, 2022).

IV.3 Generating Node similarity scores

In the embedding space, more similar nodes would have vectors that have less distance between them. Measures that determine the closeness of two vectors, such as cosine similarity and Euclidean distance, can therefore be used to generate scores for pairs of nodes. The embedding can also be used to train a logistic classifier or be used with measures of vector closeness such as cosine similarity or Euclidean distance.

Research Methods

I. Graphs used for testing

The five graphs being used for testing are:

eco-florida: A network of different ecological communities in South Florida ecosystems (Rossi & Ahmed, *Eco-florida.zip,* n.d).

inf-USAir97: A network of US Airline flights (Rossi & Ahmed, *Inf-USAir97.zip,* n.d).

LOTR: A network of hyperlinks between different character pages on the Lord of the Rings fan wiki https://tolkiengateway.net. Obtained by the author herself using BeautifulSoup, a Python web-scraping library (Richardson, 2022).

mammalia-dolphin-florida-social: A network of interactions between dolphins in Cedar Key, Florida (Rossi & Ahmed, *Mammalia-dolphin-florida-social.zip,* n.d).

road-chesapeake - an infrastructure network (Rossi & Ahmed, *Road-chesapeake.zip,* n.d).

Refer to Table 1 for the characteristics of the graphs.

Table 1. Characteristics of graphs.

Graph	Number of Nodes	Number of edges.	Average Degree
eco-florida	128	2.1K	32
inf-USAir97	332	2.1K	12
LOTR	459	7.2K	16
mammalia-dolphin-florida-social	151	1.6K	20
road-chesapeake	30	170	8

II. Graph Class

The structure used for storing data for computations is a Graph class the author coded in Python. In addition to convenience functions that find values commonly used in local node similarity indices, it also contains utility functions that return the adjacency matrix, adjacency list, and edge pair list representation of a graph. The adjacency matrix represents each undirected edge as a bidirectional directed edge. A Graph object can be initialized by a list of node pairs or read from an edge list stored in a file.

III. Sampling

Sampling is done by splitting the edge list of a graph into a train and test set using scikit-learn's train_test_split() function. Scikit-learn is a library that contains implementations of various data processing and machine learning algorithms (Pedregosa et al., 2011). For each link prediction algorithm, I will test its robustness by calculating AUC at different proportions of withheld edges.

IV. Implementation of AUC

AUC is experimentally measured by *n* random samplings. From the test set, an edge is randomly selected. Using the train set, its score is computed and compared to the score of a random nonexistent edge. The times the score of a pair of nodes forming an edge is greater than a nonexistent edge (*n'*) and the times the score is less than the nonexistent edge (*n''*) are recorded. The AUC score is then calculated according to the formula given in section **I.1**.

V. Implementation of similarity indices Random Walk with Restart and Local indices

The local similarity algorithms were implemented using Python standard library functions, according to the formulas described in the Background section of the proposal. To avoid division-by-zero errors, the value of a quotient is assumed to be zero if there is potential for a zero denominator in the proposed variation of the LHN index and the Adamic-Adar index.

The Random Walk with Restart index is implemented following the formula given in the Background using NumPy, a linear algebra library (Harris et al., 2020). Implementation-wise, I compute $M = (1 - c)(1 - cP^T)^{-1}$ instead of $\mathbf{q}_x$ to find the value of the Random Walk with Restart Index. If $M = (1 - c)(1 - cP^T)^{-1}$, then the sum M_{xy} and M_{yx} equals $q_{xy} + q_{yx}$, because multiplication by $\mathbf{e}_x$ gets the *x*-th column of the matrix *M*.

The parameter of interest is *c*. I will adjust the parameter *c* and find the AUC at each setting.

Scipy was used for reading graphs stored in the Matrix Market file format (Virtanen et al., 2020).

VI. Node2Vec prediction implementation

I will adapt Grover and Leskovec's reference implementation (2016) of Node2Vec to the custom Graph class and more recent versions of the module Gensim's Word2Vec model.

Next, I will fit a logistic regression model to the node embeddings to predict if two nodes have an edge between them. The procedure for processing embeddings, similar to the one proposed by Grover and Leskovec (2016), is as follows:

1. Given two nodes *u* and *v*, combine them using an operator, such as pairwise multiplication.
2. The resultant vector is then used as the input into the logistic regression model. The output is a value between 0 and 1, determining the probability of the pair of nodes having an edge(1) or having no edge (0).

The logistic regression model I will use is Scikit-learn's logistic regression model (Pedregosa et al., 2011). The training data set of the model will contain an even distribution of positive and negative examples. The positive examples will be the entirety of the edges not withheld for testing. The negative examples will be an equal number of randomly generated nonexistent edges. The value generated by the logistic regression classifier is treated as the score of a node pair and used to compute AUC.

I will compare the performance of the logistic regression model derived from node embedding to the performance of cosine similarity and Euclidean distance as node embedding similarity indicators.

VII. Visualization

In addition to creating a link prediction regression model using Node2Vec embeddings, I will also create visualizations of the node embeddings as a qualitative way of verifying the validity of my implementation. To create visualizations of the node embeddings, I will reduce the dimensionality of the vector embeddings to 2D using Scikit-learn's PCA model (Pedregosa et al., 2011).

VIII. Parameter of Interest in Node2Vec

The parameter of interest is the dimensionality of the embeddings. I will find and compare the AUC of the logistic regression link prediction model at different embedding dimensions.

Materials

- Python 3
- BeautifulSoup (Richardson, 2022)
- Numpy (Harris et al., 2020)
- Scikit-learn (Pedregosa et al., 2011)
- Scipy (Virtanen et al., 2020)

Conclusion

I will consider this project to be a success if I implement all aforementioned algorithms and collected the relevant data. If time permits, I will test more algorithms.

References

Bowman, B. (2022). *Graph techniques for next generation cybersecurity* (Publication 2605573371). The George Washington University. ProQuest Central; Publicly Available Content Database. https://www.proquest.com/dissertations-theses/graph-techniques-next-generation-cybersecurity/docview/2605573371/se-2

Facebook. (n.d.). *Where do People You May Know suggestions come from on Facebook?* Facebook. Retrieved November 29, 2022, from https://www.facebook.com/help/163810437015615/?helpref=uf_share

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855-864. https://doi.org/10.1145/2939672.2939754

Harris, C. R., Millman, K. J., Van der walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Van kerkwijk, M. H., Brett, M., Haldane, A., Del río, J. F., Wiebe, M., Peterson, P., . . . Oliphant, T. E. (2020). Array programming with numpy. *Nature*, *585*(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Lü, L., & Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and Its Applications*, *390*(6), 1150-1170. https://doi.org/10.1016/j.physa.2010.11.027

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., & Thirion, B. (2011). Scikit-learn: Machine learning in Python [PDF]. *Journal of Machine Learning*, *12*(85), 2825-2830. https://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf

Richardson, L. (2022). *Beautiful Soup* [Computer software]. Leonard Richardson. https://www.crummy.com/software/BeautifulSoup/

Rossi, R., & Ahmed, N. (n.d.). *Eco-florida.zip* [Dataset]. The Network Data Repository with Interactive Graph Analytics and Visualization. Retrieved November 19, 2022, from https://networkrepository.com/eco-florida.php

Rossi, R., & Ahmed, N. (n.d.). *Inf-USAir97.zip* [Dataset]. The Network Data Repository with Interactive Graph Analytics and Visualization. Retrieved November 19, 2022, from https://networkrepository.com/inf-USAir97.php

Rossi, R., & Ahmed, N. (n.d.). *Mammalia-dolphin-florida-social.zip* [Dataset]. The Network Data Repository with Interactive Graph Analytics and Visualization. Retrieved November 19, 2022, from https://networkrepository.com/mammalia-dolphin-florida-social.php

Rossi, R., & Ahmed, N. (n.d.). *Road-chesapeake.zip* [Dataset]. The Network Data Repository with Interactive Graph Analytics and Visualization. Retrieved November 19, 2022, from https://networkrepository.com/road-chesapeake.php

Vatsal. (2022, January 31). *Node2Vec explained: Explaining & implementing the Node2Vec paper in Python*. Towards Data Science. Retrieved November 19, 2022, from https://towardsdatascience.com/node2vec-explained-db86a319e9ab

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., . . . Quintero, E. A. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261-272. https://doi.org/10.1038/s41592-019-0686-2