Survey of popular Link Prediction algorithms and heuristics

Abhinav Kumar-180050003, Nimay Gupta-180050068,
Niraj Mahajan-180050069

Abstract

Link Prediction is a key problem for network structured data. It has wide practical applications in social networks, citation networks, and so on. Given a snapshot of a social network, can we infer which new interactions can be added based on the structure of the graph? This question is answered by Link prediction. In this survey, we compare the performance of various heuristics based methods and some popular supervised algorithms in link prediction.

1. Introduction

Due to the increasing surge in the research on complex graphical models, a considerable attention is devoted to Social Networks and their properties. A social network is a graph where the nodes represent entities or people, and the edges represent a link or connection between them. An example of a social network is citation graphs, where scientists working under the same discipline are the nodes, while the edges represent a collaboration.
Link Prediction is defined as: *Given a snapshot of a social network at time t, we seek to accurately predict the edges that will be added to the network during the interval from time t to a given future time t'* [9]. In other words, we need to use the intrinsic features of a social network to define a measure of proximity or similarity between any two nodes, and thus predict the probability of a link between the two.

2. Heuristic-based Methods

There exist several heuristic-based approaches to compute the similarity in the Link-Prediction problem. These heuristics are very simple, easy to implement and perform surprisingly well. Although these heuristics are not rigorously proven, work has been done on theoritically justifying the performance of these algorithms [13]
There exist many heuristic-based algorothms like Adamic-Adar, Katz Measure, Preferential Attachment, Common Neighbor, Resource Allocation, Jaccard's

coefficient. In this section, we will analyse the performance of these heuristic based methods on several sample graphs.

2.1. Common Neighbors

This is one of the most straightforward heuristic that is employed in link prediction. The score for an edge between two nodes u,v is given by simply the number of common neighboring nodes they share [11], that is,

$$score(u, v) = |\Gamma(u) \cap \Gamma(v)|$$

2.2. Preferential Attachment

Preferential Attachment has received considerable attention as a model of the growth of networks[2, 10]. The idea behind this heuristic is that the probability of a node being involved in a new edge is proportional to the degree of that node. We define $\Gamma(x)$ as the set of all neighbors of node x. Hence the score for a link between two nodes u,v is given by

$$score(u, v) = |\Gamma(u)|.|\Gamma(v)|$$

2.3. Adamic Adar

Common Neighbors can sometimes fail in the case of nodes with high degrees. For example in a Facebook social network, a lot of unrelated people follow celebrities. Adamic and Adar [1] proposed a measure similar to the common neighbors, which takes care of such a case. Instead of simply computing the number of common neighbors, we compute a weighted sum, weighing rarer features more heavily. In our case, the rarity of a feature is proportional to the degree of the node.

$$score(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{log(|\Gamma(w)|)}$$

2.4. Jaccard Coefficient

Jaccard coefficient [12] measures the probability that both x and y have a common neighbor, for a for all the neighbors that either x or y has. It is popularly known as the Intersection over Union index.

$$score(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

2.5. Katz Measure

Katz [7] defines a measure that directly sums over the collection of paths between two nodes, exponentially damped by path length.
For nodes u,v, define $\mathbf{paths}_{\mathbf{u},\mathbf{v}}^{\langle \mathbf{l} \rangle}$ as the set of all l-length paths between nodes u,v.

$$score(u,v) = \sum_{l=1}^{\inf} \beta^l . |\mathbf{paths}_{\mathbf{u},\mathbf{v}}^{\langle \mathbf{l} \rangle}|$$

where $\beta > 0$ is a dampening parameter. Using the adjacency matrix A, we can get the Katz matrix by solving the infinite geometric progression, as $\mathbf{K} = (\mathbf{I} - \beta\mathbf{A})^{-1} - \mathbf{I}$.

3. Node2vec

3.1. Random walk

In node2vec [4], given a source node u, we simulate a random walk of fixed length l. Let c_i denote the ith node in the walk, starting with $c_0 = u$. Nodes c_i are generated by the following distribution:

$$P(c_i = x | c_{i-1} = v) = \begin{Bmatrix} \frac{\pi_{vx}}{Z}, & \text{if}(v,x) \in E \\ 0, & \text{otherwise} \end{Bmatrix}$$

where π_{vx} is the unnormalized transition probability between nodes v and x

3.2. Search bias α

The simplest way to bias our random walks would be to sample the next node based on the static edge weights w_{vx} i.e., $\pi_{vx} = w_{vx}$. (In case of unweighted graphs $w_{vx} = 1$.) However, this does not account for the graph structure and also does not guide our search procedure to explore different types og neighbourhoods. We define a 2nd order random walk with two parameters p and q which guide the walk. Conider a random walk that just traversed (t,v) and resides at node v. The walk now needs to decide the next node so it evaluates the transition probabilities π_{vx} on edges (v,x) leading from v. We set the unnormalized transition probability to $\pi_{vx} = \alpha(t,x) \cdot w_{vx}$ where

$$\alpha(t,x) = \left\{ \begin{array}{cc} \frac{1}{p}, & \text{if} d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{array} \right\}$$

and d_{tx} denotes the shortest path distance between nodes t and x. Intuitively, p and q controls how fast the walk explores and leaves the neighbourhood of starting node v.

```
Algorithm 1 The node2vec algorithm.
LearnFeatures (Graph $G = (V, E, W)$, Dimensions $d$, Walks per
    node $r$, Walk length $l$, Context size $k$, Return $p$, In-out $q$)
    $\pi$ = PreprocessModifiedWeights($G, p, q$)
    $G' = (V, E, \pi)$
    Initialize $walks$ to Empty
    for $iter = 1$ to $r$ do
        for all nodes $u \in V$ do
            $walk$ = node2vecWalk($G', u, l$)
            Append $walk$ to $walks$
    $f$ = StochasticGradientDescent($k$, $d$, $walks$)
    return $f$

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node $u$, Length $l$)
    Inititalize $walk$ to $[u]$
    for $walk\_iter = 1$ to $l$ do
        $curr = walk[-1]$
        $V_{curr}$ = GetNeighbors($curr$, $G'$)
        $s$ = AliasSample($V_{curr}$, $\pi$)
        Append $s$ to $walk$
    return $walk$
```

Figure 1: Pseudocode for the Node2vec Embedding Generation Algorithm

3.3. The node2vec algorithm

The pseudocode for node2vec, is given in Figure 1. In any random walk, there is an implicit bias due to the choice of the start node u. Since we learn representations for all nodes, we offset this bias by simulating r random walks of fixed length l starting from every node. At every step of the walk, sampling is done based on the transition probabilities π_{vx} . The transition probabilities π_{vx} for the 2nd order Markov chain can be precomputed and hence, sampling of nodes while simulating the random walk can be done efficiently in O(1) time using alias sampling. The three phases of node2vec, i.e., preprocessing to compute transition probabilities, random walk simulations and optimization using SGD, are executed sequentially.

3.4. Link prediction

For the task of link prediction, we use the embeddings generated by node2vec algorithm to get the edge feature as $g(u, v) = f(u) o f(v)$ where the possible choices for binary operator o are average, element-wise product, weighted-L1, weighted L2. Then we can use some fully connected neural network for predicting the edge score. Another possible way of edge prediction could be to use the cosine similarity between the embeddings of node u and node v as the edge score

4. GraphSage

The idea behind GraphSage[5] embedding generation is to aggregate feature information from node's local neighbourhood(e.g. degrees, other node attributes) like message passing.

```
Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm
  Input  : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices
           $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$; differentiable aggregator functions
           $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$
  Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Figure 2: Pseudocode for the GraphSage Embedding Generation Algorithm

generation process in the case where the entire graph, G = (V,E) and the features of all nodes $\mathbf{x}_v \forall v \in V$, are provided as input. Kth iteration in the outer loop describes the embedding generation for kth layer. $\mathbf{h}_{N(v)}^k$ denotes the aggregated representation of the neighbourhood of node **v**. In our case, this aggregator function is Mean based i.e. the output of line4 above is the mean of node vectors of neighbours of **v**. In next step, to generate the embedding for node v, aggregated representation is concatenated with features of layer k-1 and then a fully connected layer with nonlinear activation function σ is applied on top of that.

Neighbourhood definition. In the code, we uniformly sample a fixed number of neighbours for each node rather than using all the neighbours to save on time complexity which otherwise would have been O($\|V\|$).

4.2. Learning the parameters

For learning meaningful embedding for each node, we first train the network on cora dataset for node prediction task and then after learning the embedding, we train another network on top of that for link prediction. For node classification task, loss function is the regular cross entropy loss for classification.

$$J_G(\mathbf{z}_u) = -y_u log(\sigma(\mathbf{z}_u)) - (1 - y_u) log(1 - \sigma(\mathbf{z}_u)) \tag{1}$$

4.3. Aggregator

Since a node's neighbour have no natural ordering, the aggregator function must operate over a set of vectors. We describe here the mean and pooling aggregator.

Mean Aggregator. Our first candidate aggregator function is the mean aggregator, where we simply take the elementwise mean of the vectors in the set $\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}$. The mean aggregator is nearly equivalent of convolution in image domain.

Pooling Aggregator. This aggregator is both symmetric and trainable. In this *pooling* apporach, each neighbour's vector is independently fed through a fully-connected neural network, following this transformation, an element-wise max-pooling operation is applied to aggregate information from the neighbour set:

$$AGGREGATE_k^{pool} = max(\{\sigma(\mathbf{W}_{pool}\mathbf{h}_{u_i}^k + \mathbf{b}), \forall u_i \in N(v)\}) \tag{2}$$

where max is the elementwise max operator and σ is a non-linear activation function. In principle, before max-pooling, the function applied could be arbitrarily complex but here we focus on single layer neural network

5. Evaluation

Dataset	Evaluation Method	Algorithm						
		Adamic Adar	Common Neighbor	Preferential Attachment	Katz Measure	Logistic Regression	node2vec	GraphSage
Facebook	MAP	**0.726**	0.719	0.112	0.698	0.694	0.663	————
	MRR	**0.875**	0.872	0.242	0.860	0.849	0.813	————
Celegans	MAP	0.453	**0.525**	0.304	0.446	0.453	0.400	————
	MRR	0.584	**0.653**	0.443	0.583	0.587	0.530	————
arXiv mini	MAP	0.814	**0.831**	0.077	0.759	0.212	0.741	————
	MRR	0.922	**0.936**	0.173	0.886	0.319	0.863	————
cora	MAP	0.637	**0.684**	0.169	0.500	0.466	0.405	0.01
	MRR	0.664	**0.713**	0.187	0.549	0.522	0.447	0.02

Table 1: Performance of various link predictors on several datasets. Bold entries represent the best performance.

Datasets. In our analysis we have used four datasets to compare various LP algorithms - Facebook [8], Celegans [6], arXiv [3], cora [14]. The facebook graph is social network, while arXiv and cora are citation graphs. C-elegans is a graph of neurons and synapses in C. elegans, a type of worm. we have used a reduced form of the arXiv graph (3000 nodes out of 10000).

Performance of Heuristics. We observe that Common Neighbors gives us the best results for cora, arXiv and celegans graph, while adamic adar gives the best results for the facebook graph. This is because adamic adar weights the score according to the degree of the common neighbor. Hence this shield our score from being influenced by celebrity nodes. Such nodes can be frequently occurring in the facebook graph, but will be less frequent in citation graphs as researchers have an additional hierarchies of fields, and subfields when they collaborate with others. Also, celebrity nodes in case of synapses (celegans) are also rare, since they occur at major synaptic junctions. Hence, common neighbors outperform adamic adar.
Preferential Attachment performs poorly as it is based on linking highly (socially) active nodes. This is a bad measure since it is improbable that an edge can be established between any two random nodes, just because they are socially active. For example it is absurd to say that two researchers, one in Computer Vision and one in Earth Science, will collaborate on a project.

Performance of node2vec and GraphSage. We observe that node2vec gives sub optimal results as compared to the heuristic based approaches, while GraphSage gives really bad results. Note that GraphSage requires intrinsic features as well and hence it is run only on the Cora dataset. Another observation to be noted is that the embeddings obtained using GraphSage give an 86.20% accuracy for node classification. This proves that the embeddings are sensible and contain valuable information regarding the nodes. But since the graph is extremely sparse (0.04 test fraction) the embeddings contain only the nodes' intrinsic information, and fail to extract the neighborhood information, or the edge-similarity information. This makes it difficult to train a logistic regression on the embeddings obtained from such sparse graphs.

6. Conclusion

The heuristic based methods like common neighbors, adamic adar and katz measure are simple but extremely powerful, since they gave a performance which was at par, if not better, than the supervised methods.
GraphSage does not seem to work well on extremely sparse graphs as it is not able to extract neighborhood information in such cases and has to rely on the intrinsic features of nodes.

References

[1] Lada A Adamic and Eytan Adar. "Friends and neighbors on the Web". In: *Social Networks* 25.3 (2003), pp. 211–230. ISSN: 0378-8733. DOI: https://doi.org/10.1016/S0378-8733(03)00009-1. URL: http://www.sciencedirect.com/science/article/pii/S0378873303000091.

[2] Albert-Laszlo Barabasi and Reka Albert. "Albert, R.: Emergence of Scaling in Random Networks. Science 286, 509-512". In: *Science (New York, N.Y.)* 286 (Nov. 1999), pp. 509–12. DOI: 10.1126/science.286.5439.509.

[3] Colin B. Clement et al. "On the Use of ArXiv as a Dataset". In: *CoRR* abs/1905.00075 (2019). arXiv: 1905.00075. URL: http://arxiv.org/abs/1905.00075.

[4] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks". In: *CoRR* abs/1607.00653 (2016). arXiv: 1607.00653. URL: http://arxiv.org/abs/1607.00653.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035. ISBN: 9781510860964.

[6] Marcus Kaiser and Claus C Hilgetag. "Nonoptimal Component Placement, but Short Processing Paths, due to Long-Distance Projections in Neural Systems". In: *PLoS Computational Biology* 2.7 (July 2006). Ed. by KarlEditor Friston, e95. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.0020095. URL: http://dx.doi.org/10.1371/journal.pcbi.0020095.

[7] Leo Katz. "A new status index derived from sociometric analysis". In: *Psychometrika* 18.1 (Mar. 1953), pp. 39–43. URL: http://ideas.repec.org/a/spr/psycho/v18y1953i1p39-43.html.

[8] Jure Leskovec and Julian Mcauley. "Learning to Discover Social Circles in Ego Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 539–547. URL: https://proceedings.neurips.cc/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf.

[9] David Liben-Nowell and Jon Kleinberg. "The link-prediction problem for social networks". In: *Journal of the American Society for Information Science and Technology* 58.7 (2007), pp. 1019–1031. DOI: https://doi.org/10.1002/asi.20591. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.20591. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.20591.

[10] M. Mitzenmacher. “A Brief History of Generative Models for Power Law and Lognormal Distributions”. In: *Internet Mathematics* 1.2 (2004), pp. 226–251. URL: http://www.eecs.harvard.edu/~michaelm/postscripts/im2004a.pdf.

[11] M. E. J. Newman. “Clustering and preferential attachment in growing networks”. In: *Phys. Rev. E* 64 (2 July 2001), p. 025102. DOI: 10.1103/PhysRevE.64.025102. URL: https://link.aps.org/doi/10.1103/PhysRevE.64.025102.

[12] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* USA: McGraw-Hill, Inc., 1986. ISBN: 0070544840.

[13] Purnamrita Sarkar, Deepayan Chakrabarti, and Andrew Moore. “Theoretical Justification of Popular Link Prediction Heuristics.” In: July 2011, pp. 2722–2727. DOI: 10.5591/978-1-57735-516-8/IJCAI11-453.

[14] Prithviraj Sen et al. “Collective Classification in Network Data”. In: *AI Magazine* 29.3 (Sept. 2008), p. 93. DOI: 10.1609/aimag.v29i3.2157. URL: https://doi.org/10.1609%2Faimag.v29i3.2157.