Noise-Enhanced Unsupervised Link Prediction

Reyhaneh Abdolazimi(✉) and Reza Zafarani

Data Lab, EECS Department, Syracuse University, Syracuse, USA
{rabdolaz,reza}@data.syr.edu

Abstract. Link prediction has attracted attention from multiple research areas. Although several – mostly unsupervised – link prediction methods have been proposed, improving them is still under study. In several fields of science, noise is used as an advantage to improve information processing, inspiring us to also investigate noise enhancement in link prediction. In this research, we study link prediction from a data preprocessing point of view by introducing a noise-enhanced link prediction framework that improves the links predicted by current link prediction heuristics. The framework proposes three noise methods to help predict better links. Theoretical explanation and extensive experiments on synthetic and real-world datasets show that our framework helps improve current link prediction methods.

Keywords: Link prediction · Noise-enhanced methods · Graph algorithms

1 Introduction

Link prediction is a fundamental problem in graph mining, aiming to predict the existence of a link between two nodes in a graph. Link prediction has two main tasks: (1) predicting the links that will be added to a graph in the future, and (2) identifying missing links in an observed graph. Both tasks have important applications such as in identifying interactions between proteins in bioinformatics, building recommender systems, suggesting friends in social networks, and the like.

With link prediction being useful in many applications, a large number of link prediction algorithms have been proposed, which are different in aspects such as performance (e.g., accuracy) and computational complexity. An alternative to designing a new algorithm for improving link prediction can be to modify the input data (input graph) to such an algorithm. A common approach to change data is to add noise. While noise is often redundant, and research often tries to remove or reduce its effects, it has been shown to be invaluable in many areas of science, especially in nonlinear information processing systems [3]. Noise enhancement has long been used in physical systems as *stochastic resonance* and

K. Karlapalem et al. (Eds.): PAKDD 2021, LNAI 12712, pp. 472–487, 2021.
https://doi.org/10.1007/978-3-030-75762-5_38

has also shown promise in areas such as stochastic optimization, image processing, and machine learning [2,3,15]. Such benefits of adding noise have motivated us to explore the possibility of enhancing link prediction by adding noise. Adding noise introduces an extra step to the existing algorithms. This extra *noise injection* step introduces some level of randomization to link prediction algorithms. A natural approach to introduce noise in a network is to add edges as it allows one to systematically compare the predicted links in noisy and noiseless networks.

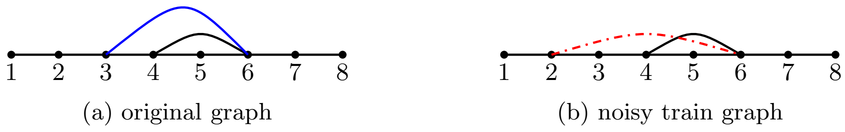

Fig. 1. Original graph before (Figure a) and after (Figure b) adding noise (dashed red edge) using the same link prediction algorithm (Adamic/Adar method). Adding noise edge $(2, 6)$ in Figure (b) improves link prediction performance, as observed by the %23 and %250 increase in the values of link prediction quantitative criteria (*ROC* and *Average Precision*, here). (Color figure online)

To provide some intuition on how adding noise can improve link prediction, we provide an example. Consider the graph in Fig. 1a with 8 nodes and 9 edges. We can split this graph into two subgraphs: (I) "train subgraph" with all 8 nodes and all the 8 black edges, and (II) "test subgraph" with 2 nodes (3 and 6) and one blue edge. We can predict the links in this graph using the Adamic/Adar link prediction method [7], and evaluate them using metrics such as *Average Precision (AP)* and *ROC*. Here, we obtain AP and ROC values of 0.14 and 0.76 respectively. Next, we add a single noise edge $(2, 6)$ to the train graph (the dashed red line) to get the *noisy train graph* in Fig. 1b. The same Adamic/Adar method can be applied to predict links. The added noise not only increases the accuracy of the predicted links (%250 increase in AP and %23 increase in ROC), but also yields a better ranking on the edges predicted (%48 increase in Kendall's Tau).

Noise-Enhanced Link Prediction. In this paper, we investigate noise-enhanced link prediction. We propose a simple framework to improve the predicted links in a network by adding noise, as outlined in Algorithm 1. Our approach first divides the original graph into two subgraphs: (1) train subgraph, and (2) test subgraph and then applies a link prediction algorithm on the train subgraph. After that, as our approach is iterative (to account for noise randomness), in each iteration, the algorithm builds a noisy network by adding noise (edges) to the train subgraph, and predict the links in this noisy network. It then evaluates the predicted links using some evaluation criteria on the original test subgraph. It iterates a few times and returns the best set of predicted links, e.g., with the best score based on some evaluation metric, as the noise-enhanced links. Our goal is to detect better links (in terms of some evaluation metric)

Algorithm 1. Noise-Enhanced Link Prediction

```
1: Input: Graph $G$, Noise Injection Method Noise, Link Prediction method LP,
          Evaluation Metric Eval, Iterations.
2: Output: Noise-enhanced ranked links $L_{\texttt{best}}$
3: $G_{\text{train}}, G_{\text{test}} \leftarrow \texttt{Divide}(G)$
4: $L_0 \leftarrow \texttt{LP}(G_{\texttt{train}})$
5: $E_0 \leftarrow \texttt{Eval}(G_{\texttt{train}},\ G_{\texttt{test}}, L_0)$
6: $E_{\text{best}} \leftarrow E_0$, $L_{\text{best}} \leftarrow L_0$
7: for $i = 1$ to Iterations do
8:     $\tilde{G}_{\text{train}_i} \leftarrow \texttt{Noise}(G_{\text{train}})$
9:     $L_i \leftarrow \texttt{LP}(\tilde{G}_{\text{train}_i})$
10:    $E_i \leftarrow \texttt{Eval}(\tilde{G}_{\text{train}_i}, G_{\text{test}}, L_i)$
11:    if $E_i$ improves compared to $E_{\text{best}}$ then
12:        $E_{\text{best}} \leftarrow E_{\text{i}}$, $L_{\text{best}} \leftarrow L_{\text{i}}$
13: return $L_{\text{best}}$
```

while adding limited noise, i.e., without extremely increasing the link prediction execution time. In particular, we aim to answer two questions:

Q1. Does adding noise improve the performance of link prediction algorithms? If yes, by how much?

Q2. Adding noise increases the cost of predicting links. Are the cost acceptable relative to the performance improvements in predicting links? What is the trade-off?

By addressing these questions at a high level, our framework makes the following contributions:

1. We introduce noise-enhanced link prediction, a framework that relies on adding noise to improve existing link prediction algorithms;
2. We propose three methods to add noise to a graph which can be translated as a preprocessing step to improve current link prediction algorithms (Sect. 3);
3. We provide a theoretical foundation for noise-enhanced link prediction by showing that the suggested noise injection methods improve link prediction measures (Sect. 5); and
4. We evaluate our framework on several real-world and synthetic networks using well-established link prediction methods. Our results show that noise helps predict better links in networks compared to the predicted links in the original graphs (Sect. 6).

2 Literature Review

To our best knowledge, there is no past research on adding noise to link prediction methods. Here, we briefly review (and relate to our work) both (I) link prediction and (II) noise-enhanced algorithms.

Link Prediction. There are two main groups of link prediction methods: unsupervised and supervised. Here, we focus on unsupervised methods. These methods attempt to predict links by assigning scores to all node pairs based on network structure. Unsupervised methods can be grouped into:

I. *Neighborhood-based Methods* assume nodes x and y are likely to link in the future if they share common neighbors. Let $\Gamma(v)$ denote the neighbors (adjacent nodes) of node v. Examples include:
   - *Common neighbors*, used in many applications [7], is formulated as $|\Gamma(x) \cap \Gamma(y)|$.
   - *Jaccard's coefficient* is the probability of selecting a common neighbor of a pair of nodes x and y among all the neighbors of nodes x and y, formulated as $\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$.
   - *Adamic/Adar* is a well-established measure for link prediction [7]. Adamic/Adar measure is defined as $\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}$, giving more weight to adjacent nodes with less neighbors.
   - *Preferential Attachment* measure assumes the probability of a new link between nodes x and y is proportional to the number of neighbors of x and y: $|\Gamma(x)| \cdot |\Gamma(y)|$

II. *Path-based Methods* consider all paths in the graph including the shortest path.
   - *Katz* counts all paths between two nodes in the graph. The paths are exponentially damped by their lengths giving more weights to shorter paths. It is measured as $\sum_{l=1}^{\infty} \beta^l \cdot |path_{x,y}^{<l>}|$, where $path_{x,y}^{<l>}$ is the set of all paths from x to y, and β is the damping parameter ($\beta, l > 0$).
   - *Hitting time* is the expected number of steps needed for a random walk to start from node x and reach node y ($H_{x,y}$). The normalized version of hitting time in undirected graphs is $NHT(x, y) = H_{x,y} \cdot \pi_y + H_{y,x} \cdot \pi_x$ where π is the stationary probability of the respective node.
   - *Rooted PageRank* is a modified version of Pagerank. In Pagerank, the score between nodes x and y can be calculated as the stationary probability of y in a random walk that moves to a random neighbor with probability β, returning to x with probability $1 - \beta$. Let D be a diagonal degree matrix, and $N = D^{-1}A$ be the normalized adjacency matrix, then $RPR(x, y) = (1 - \beta)(1 - \beta N)^{-1}$
   - *SimRank* assumes the similarity of nodes depends on the similarity of nodes that they are connected to. From a random walk viewpoint, the SimRank score measures how soon two random walkers meet at a special node if they both start from node x to y [9].

As our goal is to add noise to the link prediction process, we experiment with well-known approaches from each unsupervised link prediction category as we will discuss in our experiments.

Noise-Enhanced Systems. Noise enhances performance in many areas [3]. We review some here:

I. *Stochastic Resonance (SR)* is observed when increasing random noise leads to an increase in the signal detection performance [10]. SR is frequently used in noise-enhanced information systems with examples in biological, physical, and engineered systems [4,11].

II. *Image Processing* also benefits from noise enhancement. Adding noise to images before thresholding can improve the human brain's ability to perceive noisy visual patterns [17]. Noise can also improve image segmentation [6] and image resizing detection.

III. *Signal Detection.* Noise can help signals' detection. For example, for detecting a constant signal in a Gaussian mixture noise background, some white Gaussian noise can improve the performance of the sign detector [5]. Additive noise can also help more efficiently detect a weak sinusoid signal [21].

IV. *Optimization.* Randomization helps finding optimal or near-optimal solutions in search algorithms, when searching for an optimum is likely to get trapped in local minima. For example, the randomization in Genetic Algorithms [18] helps avoid self-similarity in the population [3]. Mutation is similar to adding noise and often a suitable mutation rate can result in performance improvement.

V. *Machine Learning.* Noise decreases the convergence time in many clustering and competitive learning algorithms [15]. It also reduces the convergence time of backpropagation algorithm while training convolutional neural network [2]. This happens as backpropagation and some clustering algorithms such as k-means which are special cases of Expectation-Maximization (EM) algorithm [16], improves by noise enhancement.

VI. *Graph algorithms.* Noise can also improve graph algorithms by modifying the input data of such algorithms. For example, it has been shown that noise can help improve current community detection methods [1] in terms of objective functions and similarity to ground-truth communities.

3 Noise Injection Methods

Based on Algorithm 1, our framework has three steps: adding noisy edges to the graph, predicting links using link prediction methods, and evaluating the predicted links via performance metrics. To analyze noise enhancement in link prediction systematically, we experiment with various link prediction methods and evaluation criteria. We propose three general ways to add noise to graphs.

Our noise injection methods focus on nodes with a high degree as links are more probable to form around higher degree nodes. This will be theoretically justified later in Sect. 5.

Therefore, we implement the following steps in the suggested noise methods : (1) sorting nodes based on their degrees, (2) choosing the top p percent of sorted nodes as *candidates*, and (3) adding edges within candidates. For adding each edge, we select a pair of nodes (edge endpoints) from candidates. The proposed methods differ in how these pairs of nodes are chosen.

I. Random Noise (`Random`). Edge endpoints are randomly selected from the candidates. Before adding a noise edge, we check its existence in the graph. If we select all nodes as candidates, `Random` simply connects nodes irrespective of their degree.

II. Weighted Noise (`Weighted`). Each node (edge endpoint) v_i is selected with probability $P_{\texttt{Weighted}}(v_i)$ that depends on its degree among degrees of other candidate nodes:

$$P_{\texttt{Weighted}}(v_i) = \frac{d_i}{\sum_{i=1}^{n} d_i} \ ,$$

where d_i refers to the degree of node i, and n is the number of candidate nodes.

III. Frequency Noise (`Frequency`). We select nodes based on the degree distribution of the candidates, where nodes with more frequent degrees are less likely to be selected:

$$P_{\texttt{Frequency}}(v_i) = \frac{1 - f_{d_i}/n}{f_{d_i} \times \sum_{d=1}^{k}(1 - f_d/n)}, \tag{1}$$

where f_d is the frequency of degree d inside candidates. This method is inspired by the observation that in most real-world networks, the degrees follow a power law distribution.

Time Complexity. Our introduced noise methods consist of the following time complexities in a graph with $|V|$ nodes and $|E|$ edges: Calculating nodes' degrees in $O(|E|)$, sorting nodes based on their degrees in $O(|V|\log|V|)$, choosing *candidates* in $O(1)$, computing node probabilities in $O(|V|)$, and adding noise edges based on node probabilities in $O(|E|)$ (The maximum number of noise edges is at most $|E|$). Hence, the final time complexity due to adding noise is $\max(O|E|, O(|V|\log|V|))$.

Example 1. Consider the graph shown in Fig. 2 with 6 nodes $\{a, b, c, d, e, f\}$ with degrees $\{1,\ 4,\ 2,\ 2,\ 1,\ 2\}$ and 6 edges. First, nodes are sorted based on their degree $\{b, c, d, f, a, e\}$, and then the top %80 of these sorted nodes are chosen as candidates $\{b, c, d, f\}$.

Finally, pairs of nodes are selected from candidates as follows:

`I. Random.` Randomly connects pairs of nodes, e.g., it may add a noise edge between nodes b and f (dashed line in Fig. 2).

`II. Weighted.` Nodes with higher degrees have higher probability to be selected. The probabilities $P_{\texttt{Weighted}}(v_i)$ for $v_i \in \{b, c, d, f\}$ are $\{0.4, 0.2, 0.2, 0.2\}$.

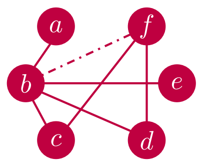

Fig. 2. Sample graph

`III. Frequency.` Candidates with less frequent degrees are more probable to be selected. So node b with degree 4 and $f_4 = 1$ is more likely ($P_{\texttt{Frequency}}(b) = 0.75$) to be chosen as a source or destination node. The frequencies of other candidates are $P_{\texttt{Frequency}}(c) = P_{\texttt{Frequency}}(d) = P_{\texttt{Frequency}}(f) = 1/12$.

Table 1. Synthetic datasets statistics

Graph model	Graph size (n)	Parameters
Random (n, p)	$1,000$	$p \in \{0.001, 0.003, 0.006, 0.007\}$
Small-world (n, k, p)	$1,000$	$p \in \{0.0001, 0.001, 0.01, 0.1, 1\}$, $k = 10$
Configuration $(deg - seq)$	$1,000$	powerlaw $deg - seq$

Table 2. Real-world datasets

(a) Real-world Datasets Statistics.

Type	**Network**	$\lvert V\rvert = n$	$\lvert E\rvert = m$	Time Range	Avg. Degree	Clustering Coeff.
Social Network	UCIrvine Messages [14]	1899	15587	2008/03-2008/07	16.4	0.109
	Bitcoin	3783	14124	2010/11-2016/01	7.46	0.176
	Internet growth [12]	25526	52412	2004/01-2007/07	4.1	0.213
	Fb Wall posts [19]	46952	876993	2004/11-2009/01	37.3	0.107

(b) Real-world Datasets details.

Network	**Train Time**	**Test Time**	#Sample Nodes	#Sample Edges
UCIrvine Messages	2008/03-2008/07	2008/08-2008/10	1899	15587
Bitcoin	2010/11-2013/12	2014/01-2016/01	3783	14124
Internet growth	2004/01-2006/12	2007/01-2007/12	8000	20310
Fb Wall posts	2004/11-2008/10	2008/11-2009/01	8000	50174

4 Experimental Setup

In this section, we describe the datasets, the proportion of noisy edges added, data preparation, candidate size, link prediction methods, performance metrics, and evaluation metrics.

I. *Datasets.* Noise impact on link prediction is studied in both synthetic and real-world networks:
(1) Synthetic Networks. To better investigate the performance of link prediction after adding noise, we evaluated our framework on synthetic graphs generated by three network models: (1) random graphs; (2) small-world model; and (3) configuration model. The properties of these three models are provided in Table 1. For random graphs, we used the *Erdős-Rényi* graph model with $n = 1,000$ nodes and edge formation probabilities equal to $\{1/n, \log n/n, 2\log n/n\}$={0.001, 0.003, 0.006, 0.007}. To create small-world graphs, we use the Watts-Strogatz model [20]. For its parameters, we use the suggestions provided by the authors [20], i.e., edge rewiring probabilities are $\{0.0001, 0.001, 0.01, 0.1, 1\}$. Finally, we use the configuration model [13] to create random graphs with specific degree sequences (we use power law).
(2) Real-world Networks: Our framework is also evaluated on real-world networks. For systematic analysis, we use four real-world networks. Table 2a provides the statistics of these networks.

II. *Data Preparation.* To ensure the data is ready for our experiments, some processing steps are taken: (1) We select data samples for each dataset. For sampling, we sample 8,000 nodes and all their connected edges (induced subgraph) using Breath-First Search. For datasets with nodes less than $8,000$ nodes, we leave them as they are. The specific sample numbers are shown in Table 2b; (2) We split the input data into training and testing sets. This division is performed based on timestamps with a ratio of training data: test data $= 90\% : 10\%$ as shown in Table 2b.

III. *Noise Proportion.* The amount of added noise (e percent) depends on the number of edges in the graph. For example, if there are 5,000 edges in the graph, we can add %10 of current edges (500 edges) as noise edges to the graph. We change e values from 1% to 10% with 1% increments.

IV. *Candidates Size.* Based on Sect. 3, the top p percent of sorted nodes are selected as candidates. We vary p from 10% to 100% with 10% increments. When $p = 100\%$, candidates contain all nodes.

V. *Link Prediction Methods.* We select four widely used unsupervised link prediction measures: (1) Common neighbor, Adamic-Adar, and preferential attachment from neighborhood-based methods, and (2) Katz from path-based methods. All selected methods have shown great performance in predicting future or missing links [7].

VI. *Performance Metrics.* Two groups of quantitative criteria can be considered for evaluating predicted links [8]: (1) *fixed-threshold* metrics, which depend on several types of thresholds, and (2) *threshold curves*, which are used when the data distribution is highly imbalanced. To evaluate the predicted links, we choose two criteria from each of the aforementioned groups; accuracy and F_1-score from fixed threshold metrics, and Receiver Operation Characteristics (ROC) and Precision-Recall (PR) from threshold curves. The Average Precision score is also used in the experiments to summarize the Precision-Recall curve. For calculating the above criteria, we use a cut-off rank k to return the top-ranked results. We vary k from 20% to 100% of test edges with 20% increments. We also evaluate *Kendall's* τ coefficient to measure the ordinal association between the ranked predicted links and test edges.

VII. *Evaluation Metrics.* To assess noise enhancement, we measure the following for each link prediction performance metric:

– `Expected First Success (EFS)` is the expected number of times that we require to add noise to the graph to ensure that we improve the predicted links at least once. For example, if we predict better links in 45 tests out of 100 tests, the expected first success is 3 as $\frac{100}{45} \simeq 2.22$. Formally, for performance metric m:

$$\texttt{EFS}_m = \left\lceil \frac{\text{number of test}}{\text{successful tests}} \right\rceil$$

– `Relative performance Improvement (RPI)` is the relative performance metric value improvement after noise enhancement:

$$\texttt{RPI}_m = \frac{m_{\text{noise-enhanced}} - m_{\text{original}}}{m_{\text{original}}} \times 100$$

Before performing the experiments, we show that these link prediction similarity-based scores can in theory be improved after adding noise.

5 Theoretical Analysis

Although empirical studies [7] show that the similarity-based measures for link prediction work well on different graphs, but there should be still ways to improve the scores. In this section, we show that all similarity-based scores can be improved by adding noisy edges between high degree nodes. To simplify, consider graph G with $|v| = n$ nodes, E edges, and two nodes i and j. Let $\Gamma(i)$ denote the set of neighbors of i and $\Gamma(j)$ the set of neighbors of j. For ease of presenting the proofs and without loss of generality, denote HighDegree as the set of nodes with higher degrees compared to that of other nodes (this can be formalized). Let $i \sim j$ denote nodes i and j are linked.

Theorem 1 (Common Neighbor Score Change). *Connecting two* HighDegree *nodes can increase common neighbor score the most.*

Proof. There are two ways to look at this. First, consider adding edges between pairs of nodes to increase the common neighbor score. How we can increase $CN(i,j)$? This increase is possible by (1) connecting neighbors x of i to j, i.e., $x \sim j$, such that $x \in \Gamma(i)$; or (2) neighbors y of j to i, i.e., $y \sim i$, $y \in \Gamma(j)$. How can we form edges to increase common neighbors of more nodes? If we select neighbors x and y such that $x, y \in$ HighDegree, more nodes find common neighbor with j and i. Similarly, if $j, i \in$ HighDegree, we will also increase the common neighbor of more nodes (i.e., i and j now act as the common neighbors). Therefore, connecting two HighDegree nodes, $x \sim j$ and $y \sim i$, increases common neighbor scores among more node pairs in G.

Secondly, we can show this property relatively compared to edges between lower degree nodes. Consider $i, j \in$ HighDegree and $i', j' \notin$ HighDegree , then $i \sim j$ increases common neighbor scores more compared to $i' \sim j'$. This is because the edge $i \sim j$ increases score for all $CN(k,j)$, $k \in \Gamma(i)$ and $CN(i,l)$, $l \in \Gamma(j)$. Similarly, edge $i' \sim j'$ increases $CN(k',j')$, $k' \in \Gamma(i')$ and $CN(i',l')$, $l' \in \Gamma(j')$. Since $|\Gamma(i)| > |\Gamma(i')|$ and $|\Gamma(j)| > |\Gamma(j')|$, $i \sim j$ can increase the common neighbor score between more nodes in G compared to $i' \sim j'$.

Theorem 2 (Adamic/Adar Score Change). *Connecting two* HighDegree *nodes can increase the Adamic/Adar score the most.*

Proof. Adamic/Adar score sums up the degree of common neighbors by giving more weight to low degree nodes and less weight to high degree nodes. Consider connecting pairs of nodes that increase Adamic/Adar score. How can connecting two nodes increase $AA(i,j)$ most? This is feasible by (1) increasing $CN(i,j)$; or (2) minimizing the degree of common neighbor nodes ($\min\{deg(k) | \forall k \in \Gamma(i) \cap \Gamma(j)\}$). As it is described in Theorem 1, connecting two HighDegree nodes increases common neighbor scores among more node pairs in G. For the second part, decreasing the degrees of common neighbor nodes is not considered in this paper as we only add edges. As adding edges decreases the weight of the common neighbor node in calculating Adamic/Adar score, how we can handle this trade-off to finally increase $AA(i,j)$?

We consider the worst-case scenario for $AA(i,j)$. If $x \in \Gamma(i)$ or $x \in \Gamma(j)$, then $x \sim k, k \notin \{i,j\}$ decreases $AA(i,j)$ by $(\frac{1}{\log(|\Gamma(x)|)} - \frac{1}{\log(|\Gamma(x)|+1)})$. This loss can be minimized if $x \in$ HighDegree since changes in logarithm value of $\Gamma(x)$ have less impact on $AA(i,j)$ while $x \in$ HighDegree compared to when $x \notin$ HighDegree. Similarly, if $k \in$ HighDegree, this weight difference will be too small. Therefore, if the gain resulted by the new common neighbors is higher than the weight loss due to current common neighbors, $AA(i,j)$ will be increased, which shows the power of connecting two high degree nodes to improve Adamic/Adar score.

Theorem 3 (Preferential Attachment Score Change). *Connecting two* HighDegree *nodes can increase the preferential attachment score the most.*

Proof. Preferential attachment selects the pair of nodes with the maximum value of $PA(i,j) = |\Gamma(i)| \cdot |\Gamma(j)|$, which indirectly implies $PA(i,j)$ mostly chooses nodes $\{i,j\} \in$ HighDegree. First, consider connecting node pairs that increase $PA(i,j)$. How does a noisy edge increase $PA(i,j)$? The gain of adding noisy edges can be calculated in two situations; 1) if we only add a noise edge $x \sim i$ ($x \notin \{i,j\}$), the gain is $|\Gamma(i)+1| \cdot |\Gamma(j)| - |\Gamma(i)| \cdot |\Gamma(j)| = |\Gamma(j)|$, and 2) if we add both noise edges $x \sim i$ and $y \sim j$ ($x, y \notin \{i,j\}$), where the gain will be $|\Gamma(i)+1| \cdot |\Gamma(j)+1| - |\Gamma(i)| \cdot |\Gamma(j)| = |\Gamma(i)| + |\Gamma(j)| + 1$. So, more neighbors in the noisy edges' endpoints lead to more gains in $PA(i,j)$, and if both $\{i,j\} \in$ HighDegree, $PA(i,j)$ can be increased more compared to the situation $\{i,j\} \notin$ HighDegree.

The mentioned gains can be extended to all sources and destinations of noisy edges, e.g., for x and y, if $\{x,y\} \in$ HighDegree, their chance of being selected by preferential attachment will be increased. Therefore, noise can also improve $PA(i,j)$ by connecting two HighDegree nodes.

Theorem 4 (Katz Score Change). *linking two* HighDegree *nodes increases Katz score the most.*

Proof. $Katz(i,j)$ counts all paths between two nodes and gives more weights to shorter paths. How we can increase $Katz(i,j)$? This increase is possible by (1) increasing the number of paths between i and j, or (2) making the path between i and j shorter.

Adding a random edge can create paths that did not exist before. This random edge can have the highest effect on $Katz(i,j)$ if it creates a path between a large number of nodes. As nodes in HighDegree act as hubs in the graph and they are connected to a large number of nodes, connecting two of them ($x \sim y$) make a bridge between $\Gamma(x)$ and $\Gamma(y)$. As a result, not only the lengths of paths between many nodes will be decreased, but also the number of paths between them will be increased. So, connecting two HighDegree nodes can also increase Katz score.

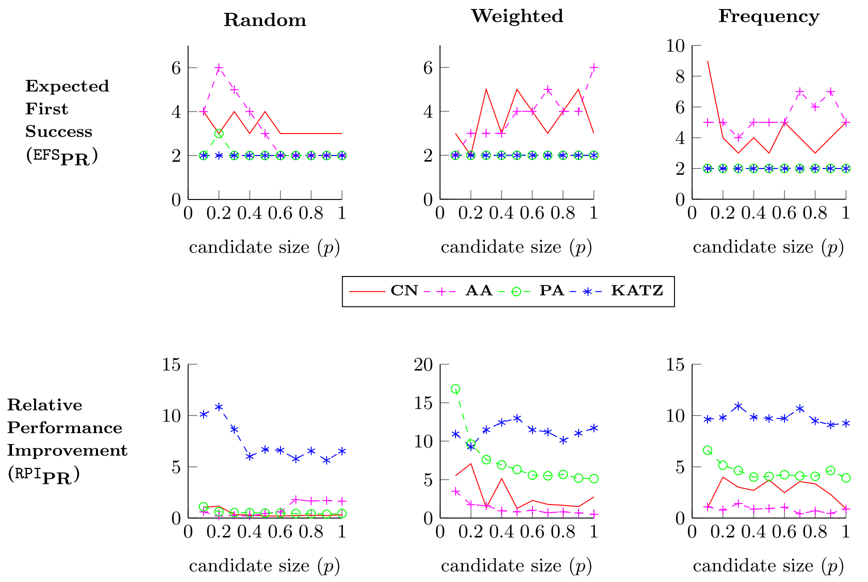

Fig. 3. Impact of all three proposed noise methods on area under precision-recall (PR) curve of the predicted links on Bitcoin. The average EFS = 3 and RPI = 4.2 for all link prediction measures.It shows we only needs to add noise three times to Bitcoin to predict better links in terms of PR.

6 Experimental Analysis

We evaluate the impact of adding noise on link prediction in both real-world and synthetic networks.

6.1 Noise-Enhanced Link Prediction in Real-World Networks

For each network and each candidate size p, we enhance the network using three noise methods by adding different proportions of noise e and measure both evaluation metrics (EFS and RPI) for all performance metrics. As e varies from 1% to 10% with 1% increments, and the experiments are done 10 times for each e (to assess stability of the results), 100 experiments are done for each p. By taking the average of EFS and RPI values of these 100 tries, the results of adding different proportions of noise for each candidate size p can be summarized by a number. The results of both evaluation metrics (RPI and EFS) on different candidate sizes, for each dataset and performance metric, can be summarized using 6 plots as shown as an example in Fig. 3. The figure shows the impact of all three proposed noise methods on the area under precision-recall (PR) curve of the predicted links on Bitcoin dataset. As shown in the figure, EFS is on average 3 for all link prediction measures, so on average, one only needs to add noise three times to Bitcoin to predict better links in terms of PR. The average RPI(= 4.2) for PR curve is also shown in Fig. 3. It is interesting that Katz works very well with noise in Fig. 3 with Low EFS (= 2) values and high RPI values.

For space reasons, we summarize all results in Table 3, which provides the average `EFS` and `RPI` of noise-enhanced link prediction methods on all real-world networks. For each network, there are six rows, one for each performance metric. For each network, noise method, and link prediction method, we provide both `EFS` and `RPI`. We summarize the findings in Table 3 as follows:

– Noise-enhanced link prediction has an average $\texttt{EFS} = 17$ and $\texttt{RPI} = 9.4$ over all networks, noise methods, link prediction methods, and performance metrics, implying noise yields improvements.
– Weighted Noise is the best noise option for Bitcoin ($\texttt{EFS} = 3$ and $\texttt{RPI} = 3.5$), and Internet growth ($\texttt{EFS} = 2$ and $\texttt{RPI} = 16$). Frequency Noise performs the best on UCIrvine Messages ($\texttt{EFS} = 4$ and $\texttt{RPI} = 29$) and Random Noise is the best noise choice for Fb Wall posts ($\texttt{EFS} = 4$ and $\texttt{RPI} = 2.3$). In overall, Weighted is the first noise priority to apply on a real-world social network; and
– Each link prediction method works best with a specific type of noise: (1) Common neighbor improves more with Frequency Noise and Weighted Noise, where for `Frequency`: ($\texttt{EFS} = 5, \texttt{RPI} = 2.8$), and for `Weighted`: ($\texttt{EFS} = 5, \texttt{RPI} = 1.8$). These numbers are the average values for the area under curve of ROC, accuracy, F_1-score, Precision-Recall, average-precision, and Kendal's tau; (2) Adamic/Adar improves more with Frequency Noise and Random Noise, where for `Frequency`: ($\texttt{EFS} = 4, \texttt{RPI} = 7.1$), and for `Random`: ($\texttt{EFS} = 7, \texttt{RPI} = 7.2$); (3) Preferential-Attachments improves more with Random Noise and Weighted Noise, where for `Random`: ($\texttt{EFS} = 3, \texttt{RPI} = 10.7$), and for `Weighted`: ($\texttt{EFS} = 7, \texttt{RPI} = 11.9$); (4) Katz improves best with Weighted Noise, where for `Weighted`: ($\texttt{EFS} = 3, \texttt{RPI} = 23.1$).

Table 3. Expected First Success (`EFS`) and Relative Performance Improvement (`RPI`) of noise-enhanced link prediction methods on 4 real-world networks. These numbers show that noise-enhanced link prediction has an average $\texttt{EFS} = 17$ and $\texttt{RPI} = 9.4$ over all networks, noise methods, link prediction methods, and performance metrics.

Network Dataset	**Performance Metric**	**Random Noise**								**Weighted Noise**								**Frequency Noise**							
		CommonNeighbor		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*	
		EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI
IrvineMessages	ROC-AUC	5	1.7	3	4.3	4	2	3	7.3	2	9.7	2	14.5	2	4.2	6	13.1	2	9.5	2	15	2	2.5	5	35.5
	AveragePrecision	4	3.4	3	5.8	5	1.2	3	41.32	29		2	33.9	2	14.2	6	62	2	28	2	35.7	2	17.9	5	93.1
	Acurracy	11	8.3	19	4.4	17	6.2	32	2.2	8	11.9	8	11.8	9	12.6	15	5.3	4	28	5	20.5	6	10.9	8	11.8
	PR-AUC	4	3.4	3	5.6	5	1.1	3	54.3	2	29	2	33.8	2	13.9	6	77	2	30	2	36.9	2	17.8	5	122.7
	F_1-score	11	8.3	19	4.4	17	6.2	32	2.2	8	12	8	11.7	9	12.5	15	5.2	4	27	5	20.4	6	10.8	8	11.8
	kendall's tau	4	95	3	34.6	3	3	3	5.9	2	576	2	80	2	7.7	6	7.4	2	670	2	84.7	2	0.1	5	4.2
Bitcoin	ROC-AUC	6	0.5	3	0.7	2	2.3	2	1.5	2	3	3	1.4	2	3.4	2	3.7	3	1.8	3	1	2	3.8	2	4.8
	AveragePrecision	3	0.4	3	1	2	0.6	2	7.7	4	1.8	3	1.5	2	7.7	2	12.3	5	1.3	4	1.2	2	4.7	2	10.8
	Acurracy	6	0.5	4	0.9	6	0.7	4	0.8	5	0.9	4	1.2	3	3	3	1.7	6	0.7	7	0.7	3	3.2	6	0.8
	PR-AUC	3	0.4	3	0.9	2	0.5	2	7.3	3	3	4	1.2	2	7.4	2	11.2	4	2.6	5	0.8	2	4.5	2	9.8
	F_1-score	6	0.51	4	0.8	6	0.7	4	0.7	5	0.88	4	1.2	3	2.9	3	1.6	6	0.74	7	0.6	3	3.1	6	0.8
	kendall's tau	2	5.8	3	3.8	2	88.4	2	5.3	2	19.4	3	7.4	2	125	2	12.6	3	10	3	4.8	2	155.9	2	16.3
Internet-growth	ROC-AUC	3	0.4	2	1.1	2	2.4	2	0.8	2	5.5	2	5.4	2	5	7	0.5	2	3.9	2	4.8	2	5.3	9	0.4
	AveragePrecision	2	0.45	2	0.9	2	1	2	5.8	2	35	1	44	2	32.9	3	9.2	2	16.5	2	20.7	2	10	4	6.3
	Acurracy	8	0.37	4	1	3	2.4	5	0.5	2	8.3	2	8.2	2	10.4	2	7	2	10.3	2	9.8	2	6.7	2	6
	PR-AUC	3	0.26	2	0.8	2	1	2	5.2	2	34	1	43	2	32.7	3	8.3	2	16	2	21.1	2	9.9	4	5.8
	F_1-score	8	0.36	4	1	3	2.3	5	0.5	2	7.9	2	7.9	2	9.9	2	6.7	2	10	2	9.4	2	6.4	2	5.7
	kendall's tau	3	8.1	2	24	2	27	2	24	2	96	2	162	2	75147	7	6.4	2	103	2	157	2	116	11	2.5
Facebook	ROC-AUC	2	1.2	3	0.7	2	5.8	2	1.2	3	1.4	3	0.8	2	6.8	3	2.9	3	1.6	4	0.6	2	13.9	2	13.8
	AveragePrecision	2	0.8	3	0.5	2	7.6	2	7.5	5	2.1	4	0.4	2	13.8	3	5.7	7	3.4	15	0.7	2	127	7	5.8
	Acurracy	9	0.21	6	0.43	6	2.9	9	0.26	151	0.01	11	0.24	4	5.2	23	0.12	334	0.009	239	0.01	2	30	455	0.008
	PR-AUC	2	0.85	3	0.48	2	6.5	2	5.7	4	1.3	5	0.42	2	11.7	3	4.4	7	2.8	15	0.78	2	145	8	4.1
	F_1-score	9	0.2	6	0.42	6	2.9	9	0.25	151	0.01	11	0.23	4	5.1	23	0.11	334	0.009	239	0.01	2	29	455	0.007
	kendall's tau	2	7.6	3	7	2	34.6	2	5.2	4	7.1	3	8	2	42	2	11.7	4	7.5	6	3.8	2	223	2	54

Table 4. Expected First Success (EFS) and Relative Performance Improvement (RPI) of noise-enhanced link prediction on networks generated by the configuration model with $n = 1,000$ nodes. Link prediction methods perform well on configuration graphs after adding noise with an average EFS $= 3$ and RPI $= 4.21$.

Network Dataset	Performance Metric	Random Noise								Weighted Noise								Frequency Noise							
		CommonNeighbor		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*	
		EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI
$n = 1000$	ROC-AUC	3	1	2	2	4	0.83	2	3.85	2	2.7	2	2.57	3	0.84	2	5.7	3	2	2	2.4	3	0.94	2	7
	AveragePrecision	3	1.23	2	3.36	5	0.33	2	7.16	2	6.7	2	7.08	2	2.89	2	11.7	2	4.57	2	6.78	2	3.4	2	10.91
	Acurracy	3	2.19	3	2.11	3	1.83	3	2.23	3	2.33	2	3.77	2	3.24	2	4	3	2.36	3	2.68	2	2.79	3	2.46
	PR-AUC	2	1.73	2	4.41	5	0.43	2	6.9	2	7.94	2	8.73	2	3.4	2	11.28	2	5.49	2	8.57	2	3.99	2	10.46
	F_1-score	3	1.92	3	1.83	3	1.52	3	1.96	3	2	2	3.25	2	2.7	2	3.56	3	2.04	3	2.31	2	2.3	3	2.16
	kendall's tau	3	3.23	2	7.37	4	2.81	2	4.4	2	8.01	2	9.91	3	3.2	2	6.73	3	4.79	2	8.83	3	3.41	2	9.3

Table 5. Expected First Success (EFS) and Relative Performance Improvement (RPI) of noise-enhanced link prediction methods on random networks with edge formation probability $p \in \{0.006, 0.007\}$. The noise-enhanced link prediction obtains the average EFS $= 71$ and RPI $= 19.2$ over all measures and noise methods on random graph models with $n = 1000$ nodes.

Network Dataset	Performance Metric	Random Noise								Weighted Noise								Frequency Noise							
		CommonNeighbor		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*		*CommonNeighbor*		*Adamic/Adar*		*PreferentialAttach*		*Katz*	
		EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI
$p = 0.006$	ROC-AUC	23	5E-5	23	1.59			6	5.21	30	4E-3	72	0.2			7	4.24	28	3E-3	112	0.21			16	1.9
	AveragePrecision	23	4E-5	56	1.93			6	53.83	30	3E-5	72	0.52			7	70.83	28	3E-5	201	0.45			16	15.31
	Acurracy	23	3E-5	77	0.8			63	1.1	30	2E-5	72	0.4			59	0.7	28	2E-5	201	0.2			201	0.2
	PR-AUC	23	2E-5	56	2.61			6	55.39	30	1E-5	72	0.96			7	95.51	28	1E-5	201	0.64			16	15.77
	F_1-score	23	6E-5	77	0.8			63	1.1	30	4E-5	72	0.4			59	0.7	28	4E-5	201	0.2			201	0.2
	kendall's tau			19	1.24			6	32.15			126	0.25			8	26.02			167	0.26	-	-	17	12.12
$p = 0.007$	ROC-AUC	5	6.03	59	1.08			2	59.92	91	0.21	251	1.32			2	67.66	9	2.66	251	0.03			2	91.82
	AveragePrecision	36	1.76	53	0.49			2	107.19	21	12.38	143	0.37			2	90.34	48	0.48	251	0			2	204.64
	Acurracy	51	1.75	91	1.1			36	2.9	21	4.3	143	0.7			44	2.6	77	1.2	501	0.2			51	2
	PR-AUC	46	1.28	51	0.76			2	116.44	20	126.51	126	0.65			2	95.55	72	0.28	167	0.04			2	237.87
	F_1-score	51	1.74	91	1.1			36	2.89	21	4.28	143	0.7			44	2.59	77	1.2	501	0.2			51	1.99
	kendall's tau	5	203.18	59	0.1			2	52.59	112	3.19	251	0.64			2	62.47	9	21.12	251	0.02			2	67.58

Table 6. Expected First Success (EFS) and Relative Performance Improvement (RPI) of noise-enhanced link prediction methods on small-world networks with edge rewiring probability $p = 1$. The average (EFS, RPI) of the introduced framework is (5,54) for all measures and noise methods in small-world graphs with $n = 1000$ nodes.

Network Dataset	Performance Metric	Random Noise								Weighted Noise								Frequency Noise							
		Adamic/Adar		*CommonNeighbor*		*Katz*		*PreferentialAttach*		*Adamic/Adar*		*CommonNeighbor*		*Katz*		*PreferentialAttach*		*Adamic/Adar*		*CommonNeighbor*		*Katz*		*PreferentialAttach*	
		EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI	EFS	RPI
$p = 1$	ROC-AUC	4	2.2	4	9.82	3	39.33	2	44.51	21	2.3	4	9.9	3	42.94	2	44.93	8	3.54	4	11.87	3	0.03	2	42.33
	AveragePrecision	5	11.03	4	27.36	3	38.67	2	285	5	26.43	4	24.37	3	42.48	2	194	4	40.44	6	19.88	3	2.E-4	2	192
	Acurracy	6	3.82	6	12.3	3	9.5	5	19.2	5	13.55	6	10.4	3	13	4	21.5	5	16.1	8	8.35	3	1E-4	5	19.55
	PR-AUC	6	134.6	4	25.52	3	45.6	2	370	5	264	4	21.97	3	54.35	2	235	4	60.31	6	19.7	3	1E-4	2	194
	F_1-score	6	3.8	6	12.24	3	9.4	5	19.09	5	13.49	6	10.35	3	12.96	4	21.38	5	16.8	8	8.3	3	2E-4	5	19.4
	kendall's tau	8	49	4	68	9	11.6	2	231	9	33	4	74.8	8	13.1	2	209	10	47.7	4	74.8	-	-	2	209

6.2 Noise-Enhanced Link Prediction in Synthetic Networks

This section evaluates noise-enhanced link prediction on synthetic graphs generated by different graph models to address the following questions: (1) which link prediction measure works best under noise?; (2) which noise-enhanced measure performs the best for each network model? (3) Which type of network model in general yields better results? Which one has the worst results? To answer these questions, we provided the results in two parts: 1) Tables including statistics on EFS and RPI values after adding noise; and 2) Tables qualitatively summarizing the results of all measures and noise methods. We explain these two parts in detail.

Quantitative Tables. For space reasons, we summarize synthetic networks' results in Tables 4, 5, and 6, which provides the average EFS and RPI of noise-enhanced link prediction methods on configuration model, random model, and small-world model graphs. For each network, noise method, and link prediction method, we provide both EFS and RPI of each performance metric. Gray cells indicate that the specific link prediction method did not improve with the specific noise method. We summarize the findings in Table 4, 5, 6 for each model as follows:

I. Configuration Model. Table 4 provides the results of noise-enhanced link prediction as follows:

– Noise-enhanced link prediction obtains an average $\texttt{EFS} = 3$ and $\texttt{RPI} = 4.21$ over configuration model networks, link prediction methods, noise methods, and performance metrics, indicating that noise is always able to predict better links in configuration model graphs.
– Although Weighted Noise is the best noise method for improving the link prediction in configuration model (average $\texttt{EFS} = 3$ and $\texttt{RPI} = 5.18$), `Frequency` and `Random` also perform very well (`Random`: $\texttt{EFS} = 3$ and $\texttt{RPI} = 2.78$, and `Frequency` $\texttt{EFS} = 3$ and $\texttt{RPI} = 4.68$).
– Katz works better under noise ($\texttt{EFS} = 3$, $\texttt{RPI} = 6.23$). Noise can also improve preferential attachment ($\texttt{EFS} = 4, \texttt{RPI} = 2.27$), but works worse than other measures. Adamic/Adar and common neighbor also show good performance after adding noise ($\texttt{EFS} = 3, \texttt{RPI} = 3.46$).

II. Random Graphs. Table 5 represents the results of our noise-enhanced framework as follows:

– For random graphs, our framework obtains the average $\texttt{EFS} = 71$ and $\texttt{RPI} = 19.2$ over all measures and noise methods when the edge formation probability $p \in \{006, 0.007\}$. When $p \in \{0.001, 0.003\}$, the noise-enhanced framework poorly works.
– Random Noise works the best with link prediction measures ($\texttt{EFS} = 37, \texttt{RPI} = 20$), and Frequency Noise works the worst for link prediction measures ($\texttt{EFS} = 113, \texttt{RPI} = 18$).
– Katz shows the best performance after adding noise ($\texttt{EFS} = 30, \texttt{RPI} = 46$) and Adamic/Adar shows the worst ($\texttt{EFS} = 147, \texttt{RPI} = 0.6$). Common neighbors also improves ($\texttt{EFS} = 36, \texttt{RPI} = 10.9$).

III. Small-world model. Table 6 represents the results of noise-enhanced framework as follows:

– The average ($\texttt{EFS}$, $\texttt{RPI}$) is (5,54) when edge rewiring probability p equals 1 in small-world graphs for all measures and noise methods.
– When $0.01 <= p <= 0.1$ (sweet spot), only Frequency Noise can perform well ($\texttt{EFS}$, $\texttt{RPI}$)$=(2, 0.03)$. `Weighted` and `Random` perform poorly. For $p = 1$, all the noise methods work well (`Random`$=$($\texttt{EFS} = 5, \texttt{RPI} = 61$), `Weighted`$=$($\texttt{EFS} = 6, \texttt{RPI} = 58$)), `Frequency`$=$($\texttt{EFS} = 5, \texttt{RPI} = 41$)).
– When $0.01 <= p <= 0.1$ (sweet spot), Adamic/Adar works the best after adding noise ($\texttt{EFS} = 2, \texttt{RPI} = 0.08$), and Katz works the worst ($\texttt{EFS} = 2, \texttt{RPI} = 0.007$). Common neighbors and preferential attachment with the average ($\texttt{EFS} = 2, \texttt{RPI} = 0.06$) also perform well with noise-enhancement. For $p = 1$, Katz performs the best after noise with the average $\texttt{EFS} = 3, \texttt{RPI} = 132$.

Qualitative Table. Tables 7a and 7b summarize all statistics and figures. Table 7a answers the question on the type of noise that works best under each network model. As shown in Table 4, the configuration model works very well with all types of noise; noise should be ordered as `Weighted`, `Frequency`, and `Random` for best performance. The noise method performing best for small-world graphs is different for edge rewiring probability $p = 0.1$ and $p = 1$. When $p <= 0.1$, only `Frequency` performs well and using `Weighted`, and `Random` is not recommended. For $p = 1$, best methods can be ranked as `Random`, `Frequency`, `Weighted`. Note that `Frequency`, and `Weighted` are both our second priority as they are not very different. For random graphs, the noise ranking is: `Random`, `Weighted`, and `Frequency`. Table 7b answers the question on the type of noise-enhanced link prediction measures that work best under each type of graph. Although all measures work well for the configuration model, Katz works slightly better. Link prediction measures' ranking on noisy Random graphs is Katz, Common neighbor, and Adamic/Adar. Katz also works the best on noisy small-world graphs with $p = 1$, but performs the worst when $p < 0.1$. According to this table, Katz works well on all noisy graph models except for small-world networks' sweet-spot.

Table 7. Synthetic networks results

(a) Which type of noise works best for each network model?

Models	Parameters	Noise Priority Ranking		
Random(n, p)	all p	Random(1^{st})	**Weighted**(2^{nd})	Frequency(3^{rd})
Small-world	$p <= 0.1$	Frequency(1^{st})	**Weighted**(2^{nd})	Random(3^{rd})
	$p = 1$	Random(1^{st})	**Frequency**(2^{nd})	**Weighted**(2^{nd})
Configuration	Powerlaw	Weighted(1^{st})	**Frequency**(2^{nd})	Random(3^{rd})

(b) Which noise-enhanced link prediction measures works best for each type of graph?

Models	Parameters	Noise-enhanced link prediction methods Ranking			
Random(n, p)	all p	Katz(1^{st})	**CN**(2^{nd})	AA(3^{rd})	-
Small-world	$p <= 0.1$	AA(1^{st})	**CN**(2^{nd})	PA(3^{rd})	Katz($4^t h$)
	$p = 1$	Katz(1^{st})	**PA**(2^{nd})	CN(3^{rd})	AA($4^t h$)
Configuration	Powerlaw	Katz(1^{st})	**AA**(2^{nd})	CN(3^{rd})	PA($4^t h$)

7 Conclusions

We proposed a framework to improve link prediction by adding noise to networks. The noise-enhanced framework adds a preprocessing phase before applying the existing link prediction methods to modify the input network. We introduced three noise methods to add noisy edges to the graph by considering nodes with high degrees as noisy edges' endpoints. Both theoretical and experimental results show that our framework can help the current link prediction methods predict better links in terms of performance metrics in both real-world and synthetic datasets.

References

1. Abdolazimi, R., Jin, S., Zafarani, R.: Noise-enhanced community detection. In: Proceedings of the 31st ACM Conference on Hypertext and Social Media, pp. 271–280 (2020)

2. Audhkhasi, K., Osoba, O., Kosko, B.: Noise-enhanced convolutional neural networks. Neural Netw. **78**, 15–23 (2016)
3. Chen, H., Varshney, L.R., Varshney, P.K.: Noise-enhanced information systems. PIEEE (2014)
4. Gammaitoni, L., Hänggi, P., Jung, P., Marchesoni, F.: Stochastic resonance. Rev. Modern Phys **70**(1), 223 (1998)
5. Kay, S.: Can detectability be improved by adding noise? IEEE Signal Proc. Lett. **7**(1), 8–10 (2000)
6. Krishna, O., Jha, R.K., Tiwari, A.K., Soni, B.: Noise induced segmentation of noisy color image. In: 2013 NCC, pp. 1–5, February 2013
7. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. J. Am. Soc. Inform. Sci. Technol. **58**(7), 1019–1031 (2007)
8. Lichtnwalter, R., Chawla, N.V.: Link prediction: fair and effective evaluation. In: 2012 IEEE/ACM ASONAM, pp. 376–383. IEEE (2012)
9. Lü, L., Zhou, T.: Link prediction in complex networks: a survey. Phys. A **390**(6), 1150–1170 (2011)
10. McDonnell, M.D., Abbott, D.: What is stochastic resonance? definitions, misconceptions, debates, and its relevance to biology. PLoS Comp. Bio. **5**(5) (2009)
11. McDonnell, M.D., Ward, L.M.: The benefits of noise in neural systems: bridging theory and experiment. Nat. Rev. Neurosci. **12**(7), 415 (2011)
12. Mislove, A.: Online social networks: measurement, analysis, and applications to distributed information systems. Ph.D. thesis, Rice University, Department of Computer Science, May 2009
13. Newman, M.E.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
14. Opsahl, T., Panzarasa, P.: Clustering in weighted networks. Soc. Netw. **31**(2), 155–163 (2009)
15. Osoba, O., Kosko, B.: Noise-enhanced clustering and competitive learning algorithms. Neural Netw. **37**, 132–140 (2013)
16. Osoba, O., Mitaim, S., Kosko, B.: The noisy expectation-maximization algorithm. Fluct. Noise Lett. **12**(03), 1350012 (2013)
17. Simonotto, E., Riani, M., Seife, C., Roberts, M., Twitty, J., Moss, F.: Visual perception of stochastic resonance. Phys. Rev. Lett. **78** (1997)
18. Tang, K.S., Man, K.F., Kwong, S., He, Q.: Genetic algorithms and their applications. IEEE Signal Process. Mag. **13**(6), 22–37 (1996)
19. Viswanath, B., Mislove, A., Cha, M., Gummadi, K.P.: On the evolution of user interaction in Facebook. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN 2009), August 2009
20. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks. Nature **393**(6684), 440 (1998)
21. Zozor, S., Amblard, P.O.: On the use of stochastic resonance in sine detection. Signal Proc. **82**(3) (2002)