Robustness in Biomedical Link Prediction

William Howard-Snyder
Department of Computer Science
University of Washington
Seattle, WA 98103
howarwil@cs.washington.edu

Pallavi Banerjee
Information School
University of Washington
Seattle, WA 98103
pallavib@uw.edu

Therese Pena Pacio
Department of Computer Science
University of Washington
Seattle, WA 98103
tpacio@cs.washington.edu

Hans Easton
Department of Computer Science
University of Washington
Seattle, WA 98103
hans00@cs.washington.edu

Abstract

In this project, we explore a variety of methods for link prediction on large, noisy, and incomplete graphs with the goal of understanding how sensitive these methods are to different forms of noise. We identify two primary types of noise: incorrect information and inaccurate information. We simulate the former by adding fake edges, and the latter by removing true edges. We also study the average case performance and the worst-case performance. We find that all models suffer from noise, but that neural-network-based models are especially sensitive to noise, in the worst-case.

1 Introduction

Graphs are used to model a wide variety of complex systems ranging from social networks with billions of users, to protein-drug interactions that are vital for understanding the effects of a treatment.

Link prediction is the task of predicting whether two nodes should have an edge between them. There are a variety of methods for predicting links, but in this work, we focus on network-based link prediction, which only uses features of the network (i.e., a node's relationship to its neighbors) to make this prediction, as opposed to node-specific attributes. Link prediction sees applications in many domains due to the ubiquity of graph data. For social networks, in which the nodes are people and edges indicate friendship, link prediction techniques are used to recommend new friendships to users. For protein-protein interaction networks, link prediction can be used to discover new relationships between proteins. However, often our knowledge about the relationships between components of these systems is inaccurate and incomplete. For example, a user's negligence of their social media profile might lead to outdated friendships; the cost of wet lab experiments prevent biologists from exploring all combinations of protein-protein interactions.

Given the importance of link prediction in the big data era, it is imperative to not only understand methods for this task, but also when they break and why they're brittle. The driving question of this project is: **How does noise and incompleteness impact link prediction?**

In this project, we explore methods for link prediction and examine how these methods are affected by varying degrees of noise. We focus on biomedical networks due to their importance in generating scientific insight, as well as the noise that is inherent in the data collection process for these networks.

Preprint. Under review.

We divide this report into sections. In Section 2, we formally describe the task of link prediction and motivate our analyses in the context of biological networks. In Section 3, we describe the methods for link prediction that we will be evaluating, as well as previous work that has analyzed robustness of link prediction methods. In Section 4, we describe the data that we use for our experiments and how it was collected. In Section 5, we describe the experiments that we performed, and report the performance of each method. In Section 6 we discuss our results, including our interpretation of them and elaborate on limitations of our analysis.

2 Preliminaries

In link prediction, we are given a graph that represents some underlying network, with a certain fraction of its edges removed and we would like to predict which edges are missing. Formally, we are given an unweighted, undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges between them. The goal is to find $G^* = (V, E^*)$, where $E \subseteq E^*$. Typically, link prediction methods are evaluated by randomly removing $\approx 10\%$ of the edges from a graph G^* to create the training graph G. The method of interest is trained on G, and assigned a score according to how well it predicts the missing links $E^* \setminus E$ as is done in [2, 12].

However, evaluating methods in this way does not reflect a method's robustness. Often the data for constructing the graph, G, is incomplete and inaccurate leading to missing and spurious links. This is especially true of biological networks like food webs, metabolic networks, and protein-protein interaction networks, where laboratory experiments are used to determine if a link exists [6]. Such experiments are costly, which prevent us from exploring all possible interactions, and there may be environmental factors that lead us to conclude there is a relationship when one is not present. To derive new insights about the interactions of biological entities through link prediction, we must first determine which methods can still perform well with incomplete and inaccurate information.

3 Relevant Methods

In this section, we present several methods that are used for link prediction and describe other analyses related to measuring robustness.

3.1 Neighborhood Similarity

Neighborhood approaches operate under the assumption that vertices will be more likely to have a link between them if there is large overlap in their sets of neighbors. Let $\mathcal{N}(x)$ be the neighbors of x, and $\texttt{score}_{CN}(x, y)$ denote common neighbor score, and $\texttt{score}_{AA}(x, y)$ denote Adamic-Adar similarity. Then, we define

$$\texttt{score}_{CN}(x, y) := |\mathcal{N}(x) \cap \mathcal{N}(y)| \qquad \texttt{score}_{AA}(x, y) := \sum_{z \in \mathcal{N}(x) \cap \mathcal{N}(y)} \frac{1}{\log |\mathcal{N}(z)|}$$

Common neighbors is quite intuitve as it is just the number of neighbors shared by x and y. Adamic-Adar Similarity is slightly more complex, but it is very similar. Instead of taking the raw count of shared neighbors, it weights the neighbors by their rarity (i.e., two nodes sharing many neighbors of degree 2, is more meaningful than sharing neighbors that have many neighbors).

3.2 Latent Factor Model

Latent Factor Models utilize Matrix Factorization to extract latent factors that capture meaningful connections between users and items. A common approach for extracting these latent factors is Singular Value Decomposition (SVD) in which we can represent a adjacency matrix A as

$$A = USV^T$$

where the embeddings of nodes are found by projecting onto rank k of U. However, SVD is undefined for missing data, making it ill-suited for the link-prediction task. In the adjacency matrix, missing edges have a value of 0. SVD treats interprets these edges as not existing, when it is possible that these edges are missing from the graph dataset. Therefore, we employ a deep-learning based approach

[10] to learn end-to-end embeddings for each node and trains an MLP model to predict linkages based on these embeddings.

3.3 Graph Neural Networks

Recently, the research community has shifted to *learning* node similarity functions with GNNs directly from the data, rather than using hand-crafted heuristics or linear latent factors. GNNs try to learn representations of nodes that encode information about the structure of the graph. The intuition is that by leveraging the topological relationships between nodes we can get higher-quality representations that yield better performance on downstream tasks (e.g., node classification, recommendation, etc.). To compute similarity between nodes (i.e., whether an edge should be present), it is common to measure the similarity between the *hidden representation* of the two nodes.

In [10], Wu et. al discuss a wide range of Graph Neural Networks for learning node/link/graph level representations and categorize them into four overarching methods: Recurrent GNNs, Convolutional GNNs, Graph Autoencoders, Spatio-Temporal GNNs. For this project, we are primarily interested in Convolutional GNNs (ConvGNNs) as they are flexible, efficient, and naturally extend to the task of link prediction.

Spatial ConvGNNs Generalize the convolutional architecture in computer vision, which operates on grid graphs, to graphs of arbitrary structure. These networks are composed of convolutional layers that iteratively compute a nodes "representation" based on that node's features and those of its neighbors. In [11], Xu et. al formally define a framework for the kth layer

$$a_v^{(k)} = \texttt{AGG}^{(k)}\left(\left\{h_u^{(k-1)} \ : \ u \in \mathcal{N}(v)\right\}\right), \quad h_v^{(k)} = \texttt{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

where $a_v^{(k)}$ is an aggregation (e.g., sum, concatenation, mean) of node representations and $h_u^{(k-1)}$ is the representation of node u for the kth layer, which is a function (e.g., MLP) of the aggregation step. Typically, $h^{(0)}$ is one-hot encoded, or node features. Broadly, Spatial ConvGNNs all fit into this form.

One particular GNN method that has recently seen a lot of success in link prediction is GraphSAGE [3]. This method is a Spatial ConvGNN, so it fits the `AGG` and `COMBINE` scheme described above. In this work, Hamilton et. al propose sampling from a nodes neighborhood in the aggregation function, rather than aggregating across all neighbors of a node. This allows for fixed batch sizes, and make training significantly more efficient.

3.4 Evaluation Methods

While we focus on the task of link prediction, we are primarily interested in how these methods perform in noisy settings. A handful of other works have considered similar settings, and we review some of them in this section.

Noisy Link Prediction In [13], Zhang et. al argue that we should evaluate link prediction methods not only on their predictive performance in the typical evaluation scheme, but we should also measure their robustness to noise in the training graph. In their work, they explore the effects of randomly adding, removing, and swapping edges, and propose a metric called *algorithm robustness* that summarizes this effect into a single number. Formally, it is defined as

$$R = \frac{1}{|L|}\sum_{q=0}^{|L|}\frac{\text{AUC}(q)}{\text{AUC}(0)}$$

where L is the number of perturbed edges and $\text{AUC}(q)$ is the AUC of a link prediction method when q links are perturbed in observed network. Ways of perturbing edges include adding, removing, and swapping edges. We intend to perform a similar analysis, but hope to extend it by also exploring the effects of adversarially perturbing edges as well.

Adversarial Link Prediction Attacks Although our domain of interest is not particularly vulnerable to link prediction attacks (i.e., there is little incentive for obscuring links in biomedical datasets), we would like to understand robustness of link prediction methods under the worst-case

noise. Therefore, we also make an effort to study adversarial link prediction attacks, where the objective is to perturb the training graph in such a way so as to maximally decrease prediction performance of link prediction algorithms.

Given $G = (V, E)$, the adversarial link prediction problem can be framed as a situation involving two parties: a seeker and an evader. The seeker's objective is to rank all the missing links in the graph, denoted as $\overline{E}$, using a similarity index and identify the highly ranked ones that are likely to exist in the network. Meanwhile, the evader possesses a subset of links in $\overline{E}$, denoted as H, that they want to keep hidden. The evader's goal is to restructure the network in a way that minimizes the possibility of the seeker identifying those links in $\overline{E}$. It is important to note that if an edge in H drops in the similarity-based ranking of edges in $\overline{E}$, it becomes less visible to the seeker.

This task is called the *Evading Link Prediction Problem*, and finding the optimal set of adversarial edges has been proven to be an NP-complete for many of the similarity-based link prediction algorithms [8]. Two adversarial attacks have been proposed that use heuristics and run in polynomial times [8]. We employ the two attack heuristics when performing adversarial perturbation on our dataset.

The CTR Heuristic Closed-Triad-Removal (CTR) works by selecting an edge, $(v, w) \in E$, such that $\exists x \in V : ((v, x) \in E \wedge (x, w) \in H)$. This implies that $(v, x), (x, w)$ and (v, w) form a closed triad. The algorithm then removes (v, w) from the network, thereby removing the closed triad whose nodes are v, w and x; hence the name Closed-Triad-Removal. Importantly, the removal of (v, w) most likely will decrease the similarity score of (x, w).

The OTC Heuristic Open-Triad-Creation (OTC) works by adding edges to the network, unlike CTR which works by removing edges. It selects a an $(v, w) \in \overline{E}$ to be added to the network such that $\exists u \in V : (w, u) \in H$, and $\exists x \in V : ((x, u) \in E \wedge ((x, w) \in \overline{E} \backslash H)$. The addition of (v, w) forms the triad that consists of (v, w), (w, x), and (x, v). This can only decrease the similarity score of (w, u) and increase that of (x, w) and (y, v).

4 Data

For our project, we use a dataset from Open Graph Benchmark (OGB) that was compiled by researchers at Stanford University [4]. OGB is a collection of benchmark datasets that are diverse, challenging and realistic, with the goal of promoting scalable, robust and reproducible research in graph machine learning.

Specifically, we will use the `ogbl-ddi` dataset which is a homogeneous, unweighted, and undirected graph that represents the interactions between drugs. The dataset originates from the DrugBank database, which is an all-inclusive web resource that can be accessed for free and provides extensive information regarding drugs, their targets, actions, and interactions, including both FDA-approved medications and those undergoing the FDA approval process [9]. The DrugBank database has gained recognition as one of the world's most frequently utilized drug reference resource following its latest 5.0 release in 2018 and is considered highly reliable, making it a suitable choice for our link prediction experimentation.

Each node in the graph represents a drug that is either FDA-approved or an experimental drug. The edges in the graph depict the interactions between drugs and can be interpreted as a situation where the combined effect of taking two drugs together is significantly different from the expected effect when the drugs are taken individually. The graph dataset comprises of approximately 4.5×10^3 nodes and 1.3×10^6 edges, making it a suitable dataset for running various missing link prediction algorithms.

The drug samples are separated into train, test, and validation sets based on the proteins each drug targets in human body. This leads to a test set that includes drugs that bind to different proteins compared to those in the train and validation sets, meaning they have distinct biological mechanisms of action. This protein-target split allows for an assessment of the model's ability to make practical and meaningful predictions, rather than being limited by the existence of already known, similar medications used for training.

5 Experiments

In this section, we describe the experiments that we performed, our method of evaluation, and our results. The code for running the experiments and extracting the results can be found in this GitHub repository.

5.1 Description

We performed three types of experiments on each of the models. Each experiment consists of training the model on a particular training set and evaluating its performance on a fixed test set. The training datasets are summarized below:

1. **Original**: Use original training set as described in Section 4.
2. **Random**: Construct training graph by removing/adding k edges at random.
3. **Adversarial**: Construct training graph by removing/adding k "most important" edges.

Randomly *removing* k edges involves selecting k edges uniformly at random and removing those from the training data. Randomly *adding* k edges involves choosing k edges that are not in the list of edges used for training uniformly at random and adding those to the training data.

Adversarially removing and adding k edges is similar to random perturbations, except that instead of choosing edges uniformly at random, we choose edges according to which are the most "important" in the graph. To measure importance, we considered the number of triangles that would be added to the graph if that prospective edge was added, and the number of triangles that would be removed if that edge was removed. We select the edges or prospective edges that, if targeted, would remove/create the most triangles. This heuristic comes from [8], and we discuss it more in depth in Section 2.

This produces four perturbation types: random-remove, random-add, adversarial-remove, and adversarial-add. Removing or adding edges represents the graph containg incomplete and incorrect information, respectively. And, using random versus adversarial modifications represents the average case, and the worst case performance, respectively.

For each perturbation type, we create a dataset by varying the degree of perturbation of that type. We consider degrees ranging from removing 50% of the training edges to adding 100% edges that aren't in the training set (e.g., -0.5 and 1 in Table 2). Across all perturbation types and degrees, there were a total of 14 datasets including the baseline. Then, for each of the three models we trained and evaluted it on every dataset.

5.2 Evaluation

In the Related Work section we describe several methods for link prediction. We evaluate these methods, as summarized in Table 1.

Table 1: Methods

Type	Name	Description
Deep Learning	GraphSAGE (GNN) [3]	Learn node representations via 2-hop subsampled graph neighborhood.
Latent Factors	Matrix Factorization (MF) [4]	End-to-end learn node embeddings and MLP on top of those.
Neighborhood	Common Neighbors (CN) [5]	Neighbors are similar to the extent that they share neighbors.

We evaluate how well each method predicts new association edges given the training edges. Following in the footsteps of [4], we use the Hits@K metric to measure prediction performance. Specifically, we have a held out set of 100,000 edges that we removed from the original graph before training. Since these are edges that we want to predict are in the graph, we call them positive test edges. We sample N negative test edges (i.e., edges that are neither in the training graph nor the positive test

Table 2: Hits@100 in performance measured in $\%$ for each method on varying degrees of random and adversarial perturbation. We bold the best performing model for a given degree of perturbation and perturbation type. The use of a "–" indicates that we did not use this perturbation type.

		Remove			Baseline				Add
		-0.5	-0.25	-0.1	0	0.1	0.25	0.5	1
Random	GNN	**74.20**	**63.75**	**65.15**	**80.66**	**78.59**	**70.90**	**72.93**	**67.81**
	CN	30.85	34.02	34.61	34.52	32.52	30.74	27.96	23.44
	MF	17.90	27.70	34.16	47.92	44.27	40.64	37.73	30.75
Adversarial	GNN	**13.31**	**22.92**	**51.62**	**80.66**	0.59	0.50	0.01	–
	CN	4.24	11.61	31.41	34.52	**16.37**	**14.33**	**15.33**	–
	MF	0.0	4.35	20.92	47.92	0.29	0.07	0.0	–

set). Then, for each positive test edge we rank it with all the negative test edges by assigning a score indicating the confidence that each edge is in the graph. This creates a sorted list of $N + 1$ edges. Hits@K is the proportion of test edges that are ranked as one of the top K edges. In [4], Hu et. al found that $N = 100,000$ and $K = 100$ were reasonable choices for this dataset.

We also evaluate models by measuring the relative drop in performance for different degrees of perturbation aggregated across all edges. Specifically, for every edge e we consider the rank of that edge $R_p^{(M)}(e)$ under perturbation p with model M. Recall, that the rank of a positive edge e is the position it would have if it were inserted into the list of negative test edges sorted by their score. The median change in rank for model M and perturbation p is the median value of

$$\left\{ \frac{R_p^{(M)}(e)}{R_0^{(M)}(e)} \right\}_{e \in E* \setminus E} \tag{1}$$

where $E^* \setminus E$ is the held out test edges. This metric provides a clearer picture of how the individual edge scores are affected by perturbations in the training data. We use the median instead of mean value of the expression because the data points most likely contain very extreme outliers as a result of the perturbation, and median measurement is more robust to outliers compared to the mean.

5.3 Results

Random Perturbation We summarize performance of each model in Table 2. The best performing model for every degree of perturbation was GraphSAGE (GNN). In fact, no degree of random perturbation that we measured makes GNN perform worse than the baseline performance of the other two methods Common Neighbors (CN) and Matrix Factorization (MF) that we implemented. The worst Hits@100 that GNN gets is 63.75% for -0.25, which is at least 15% better than the baseline performance of 34.52% and 47.92% for CN and MF, respectively. Between MF and CN, MF performs better when edges are added, however, when removing edges, CN starts to outperform MF in the range -0.25 to -0.5. The left plot in Figure 1 depicts these results graphically. This figure clearly shows the dominant performance of GNN, as well as the perturbation ranges where MF starts to perform worse than CN.

The relative change in median rank can be seen in the right subplot of Figure 1. Here, we can see that the performance of CN is least affected by random perturbations. Removing edges has little effect on the performance, and when edges are added, the median rank of positive test edges is not increased by more than a factor of 2. The rankings of positive test edges that GNN produces are slightly more impacted by random perturbations. For removing edges, the median change in rank fluctuates between 1.1 and 1.6 meaning that edges are ranked $1.1\times$ to $1.6\times$ lower than the baseline. For adding edges, the median change in rank increase up to a factor of 2.2 for 0.5 perturbation. The model whose ranking is most effected by random perturbation is MF. At -0.25 perturbation, the model ranks positive edges $3\times$ lower than the baseline. And, at -0.5 perturbation, the model ranks positive edges $9\times$ lower than the baseline. The effect of adding edges is less pronounced, but still severe. At 1 perturbation, the model ranks positive edges about $3.5\times$ lower than the baseline.

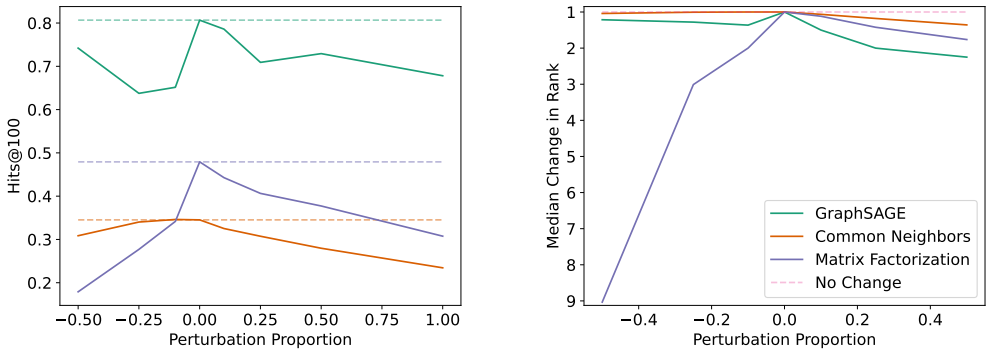

Figure 1: Plot the absolute (left) and relative (right) performance of each model for varying degrees of *random* perturbation. Negative indicates removing edges, positive indicates adding edges. In the left plot, the color-coded dotted lines indicate the performance of the baseline. In the right plot, the color-coded dotted line indicates no change in edge performance.

Adversarial Perturbation We also summarize the performance of each model for adversarial perturbations in Table 2. We can see that GNN outperforms the other two models by at least 10% for each degree of adversarial remove. However, its performance severely drops off to 0.59% for adversarial add 0.1, and gets even worse for 0.25 and 0.5. MF performs similarly poorly for each degree of adversarial additions, getting 0.29%, 0.07%, and 0% Hits@100 for 0.1, 0.25, and 0.5, respectively. The performance of CN is least affected by adversarial additions. For adversarial add 0.1 it drops by about 17% Hits@100, but roughly maintains its performance for adversarial add 0.25 and 0.5. The results are also depicted graphically in the left subplot of Figure 2.

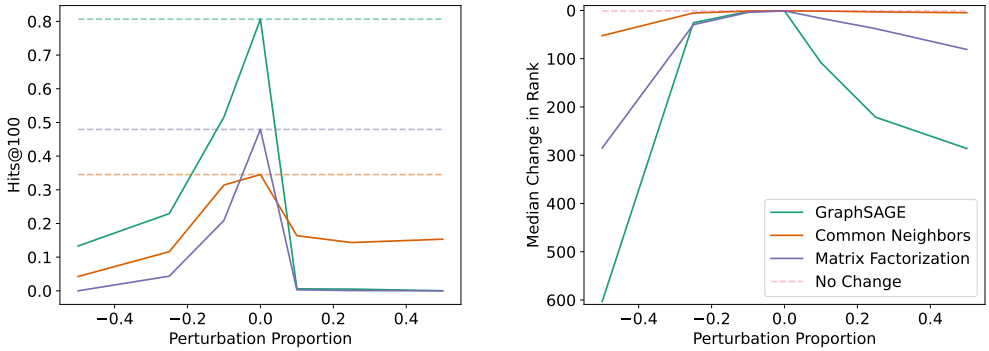

Figure 2: Plot the absolute (left) and relative (right) performance of each model for varying degrees of *adversarial* perturbation.

The relative change in median rank can be seen in the right subplot of Figure 2. Again, we see that the performance of CN is least affected by adversarial perturbations. The most significant change is for removing half the edges CN ranks positive edges $50\times$ lower than the baseline. Other than that, CN is relatively unaffected by adversarial perturbations. However, the change in edge ranking for GNN and MF is much more pronounced. For -0.25 perturbation, both models rank positive edges about $40\times$ lower than their respective baselines. And, for -0.5 perturbation GNN ranks positive edges $600\times$, and MF ranks positive edges $300\times$ worse than their baselines. The effect of adding edges on median change in edge rank is less pronounced, but still severe. For 0.1, 0.25, and 0.5 perturbation, the factor decrease in positive edge ranking of MF increases roughly linearly from $15\times$, $50\times$, and $80\times$, respectively. The change in edge ranking is more extreme for GNN. It increases to $100\times$ for 0.1 perturbation, then over $200\times$ for 0.25 perturbation, and about $280\times$ for 0.5 perturbation.

6 Discussion

Random Perturbation Our results demonstrate that GNN has the highest Hits@K performance for both randomly adding and removing edges. However, examining the Median Change in Rank graph in the right subplot of Fig 1 shows that the performance of CN is least affected by random perturbations. This finding is surprising given that GNN outperforms CN in the Hits@K metric. Previous work [1] in the sensitivity of GNNs suggests that the performance of GNNs under perturbations depends highly on the structures of the graph post-modification. These structures include node locality and topology as well as the location of the edge perturbations.

More specifically, robustness of the GraphSAGE algorithm is directly influenced by the number of edge modifications applied in the graph. Adding or deleting many edges at random in a given local neighborhood might significantly alter a local neighborhood structure. Since our GraphSAGE implementation uses $k = 2$ hops, it is relatively sensitive to local structural changes. This partially explains the overall downward trend for prediction accuracy as more edges are randomly added to the graph. Additionally, the robustness is also influenced by the degree of a node. If a node has many neighbors, changes in the edges of a single neighbor will be less noticeable during the aggregation step of the GraphSAGE node embedding learning process. Suppose there is a subset of nodes that play an important part in link prediction decisions, and the degrees of each node are relatively high. In that case, random perturbation will unlikely alter node embeddings drastically according to our reasoning earlier. This could be a possible explanation for why there is not a dramatic decrease in performance in the GraphSAGE model compared to adversarial perturbation.

Another interesting result seen in the right subplot of 1 was that GNN drops in Hits@100 after removing 25% edges, but sees an improvement after after removing and additional 25% of edges. As demonstrated by [7], removing nodes from a graph, and therefore it's incident edges, may actually improve the expressiveness of the trained models when compared to models trained on non-perturbed graphs. These lead to more distinct edge embeddings in the models, and therefore improved predictors.

Adversarial Perturbation We found that each amount of adversarial perturbation decreased the performance of every model much more than random perturbation. However, the degree to which performance was affected varied drastically across models and perturbation types.

For similarity-based link prediction algorithms, such as CN, the prediction accuracy performs slightly better when 50% edges are adversarially added (19.19%) compared to adversarially removed (30.28%). This is consistent with the findings in [8], in which the study shows that CTR-based perturbations are more effective than OTC-based perturbations. In other words, in order to hide a correct link, unlinking carefully chosen edges can provide a better disguise than adding new ones for similarity-based link prediction algorithms. In addition, the study also reveals that the attack tolerance of those similarity indices tends to decrease with average degree of nodes in the graph, especially for CTR. This finding is partially consistent with our experiment, which shows that the DDI graph we have selected is relatively dense and has a high average degree, making the network more susceptible to CTR attacks.

For neural-network based prediction algorithms, there was a significant disparity between the effect of adversarially *removing* versus adversarially *adding* edges. In Table 2, we can see that -0.1 perturbation decreases the GNN baseline's Hits@100 by less than 30%, while 0.1 perturbation decreases it by over 80%. The MF model sees a similar drop in performance of 47.92% to 0.29% Hits@100 for 0.1 perturbation. This suggests that, in the worst case, these neural-network-based models are much more sensitive to incorrect information than incomplete information. Additionally, it does not take many incorrect edges to induce large drops in performance. Determining the training data for drug-drug interactions requires wet lab experiments and causal population studies, both which can introduce error (e.g., true interactions could be obscured due to measurement error and confounding variables could lead us to think drugs interact when they don't). Having 10% of edges be incorrect is relatively small given the error-prone methods for determining interactions.

The right subplot of Figure 2, examines the relative change in performance at the edge level for each model. Here' we can see that 0.5 and -0.5 perturbation result in 300 and 600 median change in edge rank, respectively. This means that out of all of the positive test edges, half of them are ranked

300–$600\times$ lower than the baseline when half of the edges are perturbed. This change in positive test edge ranking demonstrates how vulnerable GNN is to large amounts of adversarial noise.

Although this domain is not one where we expect an adversary to poison our training graph, the adversarial framework allows us to understand the worst-case effects of noise. And in the worst-case, GNN would flag many false positives (i.e., claiming drugs that don't interact do in fact interact) before getting a true positive. Since the goal of automated drug-drug interaction prediction is to facilitate and augment doctor's ability to prescribe drugs safely, in the worst-case, these models would impede a doctor's ability to prescribe drugs. This is because it would tell them that many combinations of drugs are unsafe, when they aren't, forcing the doctor to check the combinations by some other method, or not perscribe medication when it is safe.

6.1 Limitations

There were two primary limitations we faced in this project: First, we lacked compute resources to train more models or more datasets. The second is more fundamental, and applies to any analysis like ours that tries to make a connection between perturbation and model performance.

Limited Resources We had severely limited resources for this project, especially for training the neural-network-based models. Unfortunately, none of the members had access to GPUs, so we were forced to train our models on CPUs, which could take up to 12 hours for a model-dataset pair. Due to these constraints, we could only train each GraphSAGE model for 200 epochs. Although, the validation loss pretty much converged, we would have liked to train for longer. We were also unable to do a full hyperparameter sweep and relied on values for learning rate, dropout, hidden dimensions, number of layers, etc. that other papers had found worked well in practice.

Inconsistent Evaluation The motivation for this analysis is that graphs are noisy and incomplete, and so we should make an attempt understand how are models are affected by these forms of inaccurate information. However, this suggests we should expect our original graph to also be noisy and incomplete. Therefore, the effects of perturbation that we add to the training graph may be somewhat obscured by the fact that the training graph is already noisy and incomplete. We've attempted to mitigate this effect by selecting a relatively dense graph ($\approx 90\%$ sparsity), which suggests it has little incompleteness, and that is high quality, which suggests that edges are correct. Still, the effects of each perturbation need to be taken with a grain of salt, because there could be other forms of inaccurate/incomplete information in our original training graph.

6.2 Future Work

Increase Statistical Power Given the time constraints we were only able to run our models on the perturbed datasets *once*. To make sure our claims are statistically signifcant, we would need to have many perturbed datasets for each perturbation type and degree of perturbation. This would allow us to do hypothesis tests and lend statistical power to our analysis.

Node2Vec Experimentation We have a working implementation of Node2Vec link prediction algorithm. The algorithm learns node embedding through biased random walk, and then feeds these embedding through a multi-layer neural network to predict links in the graph. For our project, we did not have enough compute power to tune the hyperparameters and derive results for all the perturbed versions of the dataset. For future work, we would like to finish the experimentation of the Node2Vec implementation and compare the robustness with other link prediction algorithms.

Finer Degrees of Perturbation We only explored perturbation at levels of 0.1, 0.25, 0.5, and 1. However, it would interesting to get a better sense of how the performance changes for the in-between values. For instance, the performance of GNN and MF dropped so suddenly for adversarially adding edges, that we couldn't get a clear picture of how it would perform under 0.01 perturbation. With enough datapoints, we could derive something analogous to scaling laws for large language models, but for robustness in link prediction.

Examination of Errors It would be very interesting to investigate what properties of edges make the models' performance drop. For example, are edges with a more central locality in the graph harder

to predict after perturbing the graph? It would also be interesting to incorporate domain specific knowledge of these drugs to identify characteristics of our nodes that may contributing to our errors. Characterizing these edges could inform how we can create more robust models

Expansion of Scope Currently, we have focused on undirected and unweighted graphs. In order to get a more well rounded idea about robustness in biomedical Link Prediction, an interesting extension will be to look at how the performance varies in case of weighted graphs.

7 Individual Contributions

Hans

- Code: Implemented *NetworkX Common Neighbor* and *Adamic Adar* similarity-based link prediction algorithms.
- Code: Implemented *random add, random remove, random swap, OTC adversarial add* and *CTR adversarial remove* perturbation implementation. Generated **12** versions of the randomized and adversarial datasets for training link prediction algorithms.
- Code: Implemented working version of *Node2Vec graph neural network algorithm*, although we did not have enough time to train and benchmark the algorithm - see 6.2 for more detail.
- Writing: Wrote section 4 on data collection, as well as a portion of 3.4 on adversarial link prediction attacks and a portion of 6 on random and adversarial analysis.
- Others: Set up GCP virtual server to train *GraphSAGE* model on adversarial add **0.1%** and adversarial remove **0.1%**.
- Others: Participated in group meetings, and attended Aniket's and Professor Althoff's office hours for project-related questions. Presented a portion of the final presentation.
- Finally, I want to give a **huge shoutout to William**, who took a leadership role throughout and contributed the most to the project. He organized the team meetings and helped the team stay on track to finish the project strong. Thanks William!

Therese

- Implemented, trained, and evaluated Matrix Factorization model on all perturbation datasets for link prediction task
- Set up GCP server to train GraphSage model on adversarial add-0.25 and remove-0.25
- Wrote about the Matrix Factorization model in the paper and the analysis of the random perturbations in the project report
- Experimented with sampling techniques to better visualize average degree vs. average change in rank (plot not included in the report)
- Participated in group meetings, office hours, and a portion of the final presentation
- I second the kudos William! Really organized the project and set up infrastructure to make delegating jobs seamless! He's the ML for Big Data GOAT and deserves a 4.0!

William

- Writing: Wrote the Abstract, Introduction, Preliminaries, and Experiments sections, as well as the subsections 3.1, 3.3, 6.1, significant portions of the sections 3.4, 6.2, and made edits throughout the report. I also typesetted the figures and tables.
- Programming: Wrote the model abstract class for organizing our model implementations (all classes implemented this interface). Implemented GraphSAGE, (Runtime) Common Neighbors, and debugged Matrix Factorization. Also, wrote the code to extract results from trained models and generate figures.
- Reading: Performed a literature review to find methods for link prediction and a second literature review to find methods for adversarial perturbations.
- Other: Set up GCP server to train GraphSAGE models on the randomly perturbed datasets, and four of the six adversarial datasets. Also, organized and lead group meetings, delegated tasks, and asked questions to our assigned TA (Aniket) and Professor to make sure we were on track.

Pallavi

- Read up on different possible ideas and literature to narrow down on GNN link prediction task and datasets

- Ran SEAL-OGB from Facebook research on OGB-DDI dataset
- Implemented initial Node2Vec model which did not perform well (Refer to Node2vec mentioned by Hans for better performance. Kudos to Hans on that! And to Therese for helping me align with the abstraction)
- Participation in group meetings and presentation
- Provided GPU access to run two adversarial add data files
- Contributed to previous written report iterations (challenges and future work)
- Last (but not the least!) huge shoutout to William for taking the lead and being extremely helpful throughout the process with maximum brainstorming!

References

[1] E. Dai, W. Jin, H. Liu, and S. Wang. Towards robust graph neural networks for noisy graphs with sparse labels. New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391320. doi: 10.1145/3488560.3498408. URL https://doi.org/10.1145/3488560.3498408.

[2] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks, 2016. URL https://arxiv.org/abs/1607.00653.

[3] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL http://arxiv.org/abs/1706.02216.

[4] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *CoRR*, abs/2005.00687, 2020. URL https://arxiv.org/abs/2005.00687.

[5] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, page 556–559, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137230. doi: 10.1145/956863.956972. URL https://doi.org/10.1145/956863.956972.

[6] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, mar 2011. doi: 10.1016/j.physa.2010.11.027. URL https://doi.org/10.1016%2Fj.physa.2010.11.027.

[7] P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks, 2021. URL https://arxiv.org/abs/2111.06283.

[8] M. Waniek, K. Zhou, Y. Vorobeychik, E. Moro, T. P. Michalak, and T. Rahwan. Attack tolerance of link prediction algorithms: How to hide your relations in a social network, 2018. URL https://arxiv.org/abs/1809.00152.

[9] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, TanvirSajed, D. Johnson, C. Li, Z. Sayeeda, and et al. Drugbank 5.0: a major update to the drugbank database for 2018. *PubMed*, jan 2018. doi: 10.1093/nar/gkx1037.PMID:29126136.

[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

[11] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

[12] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *CoRR*, abs/1802.09691, 2018. URL http://arxiv.org/abs/1802.09691.

[13] P. Zhang, X. Wang, F. Wang, A. Zeng, and J. Xiao. Measuring the robustness of link prediction algorithms under noisy environment. *Scientific Reports*, 6:18881, 01 2016. doi: 10.1038/srep18881.