Augmenting the performance of deep-learning based link prediction models with simple heuristics

Vaishnavi Gupta
vg222

Adarsh Sriram
as2566

Eleanor Goh
eg552

Abstract

Given the structure of a graph, we explore the link prediction problem of predicting future edges between nodes. We first explored the effectiveness of heuristic and neural network approaches in solving the link prediction problem. We proposed a new hybrid approach in which we combine both structural information from heuristics and representations learned by a network. Our experiments found that augmenting a baseline deep learning GCN approach with a local or global heuristic achieved statistically significant improvements, and more consistent results across datasets. The code we used to run experiments can be found here.

1 Introduction

The link prediction problem is a widely studied problem in network analysis, pertaining to predicting the existence of new edges between nodes, given a graph structure. These new edges can be considered in the context of the graph evolving over time, or filling in incomplete edges that are highly likely to exist. At a higher scale, the very general nature of this problem means that these models are applicable to a large variety of fields, like analyzing social network connections, predicting various protein and drug interactions, studying food webs, etc.

Earlier approaches to the link prediction problem involve using local and global heuristic methods [2] and probabilistic methods to analyze the probability of the existence of an edge. Such methods are based purely on the topological structure of the graph, but more recently, Graph Convolutional Networks (GCNs) have been used to utilize node feature information as well. This approach is detailed in the paper “Variational Graph Auto-Encoders" [1]. The general process involves learning embeddings for each node through a trained GCN and then using the inner product between the node embeddings as a decoding step to predict the existence of an edge. There are also newer methods with more complicated augmentations [4], but they still use GNN learning as a final step, and the VGAE method remains a good baseline.

With an explosion of available graph data, it has become easier to test the performance of these models on different types of graph structures. With their strong ability to learn

complex patterns in data, neural networks are the current state-of-the-art, but the GCN approach can also have limitations and biases, given certain types of data.

We first benchmark the performance of heuristic and GCN approaches on two Open Graph Benchmark (OGB) datasets. Analyzing the performance of these methods relative to the structure of the datasets, we describe the potential pitfalls of both heuristics and GCNs. Finally, we suggest a simple method to augment the GCN model using known heuristics. Our hybrid approach, which performs well more consistently, involves augmenting representations of pairs of nodes learned by the GCN by either a local or a global heuristic. In our experiments, we used the Adamic/Adar Index as our local heuristic and Katz scores of pairs of nodes as our global heuristic.

2 Overview of Link Prediction Methods

We focus on two core approaches to the link prediction problem - heuristics and GCNs. In this section, we provide a technical overview of the main methods we will be analyzing and testing later.

2.1 Similarity Based Heuristic Approach

Given a (potentially directed) graph $G = (V, E)$, the edges E represent the connections between nodes at a given point in time. Given all the pairs of nodes (x, y) in G that don't have an edge between them, the goal is to produce a ranked list of edges that are predicted to appear in the graph. This can be interpreted as either edges appearing a future time-step, or filling in missing edges.

Using a chosen heuristic, we define a `Score(x, y)` value, and rank all the pairs in descending order to get a sorted list of most likely edges. There are two broad categories of heuristics that can be used for the score - global and local.

2.1.1 Local Heuristics

Perhaps the simplest approach is to consider the shortest path distance between x and y. The problem with this approach is that in large connected graphs with the "small-world" property, there are many nodes at a distance of 2 away from each other, and this criterion doesn't help us differentiate between pairs.

A more refined approach is calculating scores based on the local neighbourhoods of nodes x and y, with the intuition that edges are highly likely to form between nodes with overlapping immediate neighbourhoods. We use the notation $\Gamma(x)$ to denote the neighbours of x in G. Some of the most popular heuristics in this area include

- Common neighbours: `Score(x, y)` = $|\Gamma(x) \cap \Gamma(y)|$

  CN computes the number of shared neighbours between the two nodes.

- Adamic/Adar Index:

$$\texttt{Score(x, y)} = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log \Gamma(y)}$$

An improvement upon common neighbours, this measure aims to look at all the common neighbours of x and y, and weight them by "rarity" of feature. It is more likely that x and y are connected if they are connected to a lesser connected node, as opposed to if they share a highly connected neighbour (since it is likely that many different types of nodes could share a highly connected neighbour).

2.1.2 Global Heuristics

In Liben-Nowell and Kleinberg, 2004, they analyzed that less intuitive heuristics that took into account global structure did better more consistently than local heuristics. The Katz statistic is a very popular global heuristic that is used.

- Katz: This measure sums over the set of paths between two nodes, and is exponentially damped, to give shorter paths more weight. It is given by:

$$katz(x, y) = \sum_{l=1}^{\infty} \beta^l \times |paths_{(x,y)}^{(l)}|$$

where β is the dampening parameter, and $paths_{(x,y)}^{(l)}$ is the set of all l-length paths between nodes x and y.

2.2 Variational Graph Auto-Encoders (GCNs)

Graph convolutional networks (GCNs) are used to learn single-node representations, by aggregating the features of neighboring nodes as well as the node itself, for each node in a graph, with the depth of the network being the number of hops considered for feature aggregation. VGAE applies a GCN to a network, to obtain the learned representation for each node. Then, the representation for the source and destination node for each edge are aggregated to predict a target link, thus achieving multi-node representations for the network's node set.

Given a network with n nodes with adjacency matrix $\mathbf{A}$, with node feature matrix $\mathbf{X}$ (feature vectors of dimension d), the product $\mathbf{AX}$ gives us the sum of feature values for the neighbors of each node. To include the feature vector of a node itself in the aggregation of neighbor features, we can consider the adjacency matrix with self-loops added, i.e. $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Let D be the degree matrix of $\hat{\mathbf{A}}$. It can be shown that $D^{-1/2}\hat{\mathbf{A}}D^{1/2}\mathbf{X}$ gives us the average of neighboring features (including each node itself) for each node, scaled by row and column. This scaling gives us the weighted average, with larger weights on low-degree nodes. If we add an activation function to the resultant vector, these operations represent a single layer of a GCN. For a scaled adjacency matrix $\mathbf{S} = D^{-1/2}\hat{\mathbf{A}}D^{1/2}$, the ReLU activation function, and learnable weight matrices $W^{(0)}$ and $W^{(1)}$, a 2-layer GCN will take the form:

$$f(X, A) = \text{ReLU}(S(\text{ReLU}(SXW^{(0)})W^{(1)}))$$

In the VGAE paper, given these node embeddings, the final aggregation step to learn multi-node representations was done using a simple inner-product decoder, giving us

$$\hat{A}_{x,y} = \sigma(z_x \cdot z_y)$$

Here, $\hat{A}_{x,y}$ represents the chance of there being an edge between x and y.

In practice, for the aggregation function, an improvement would be to use a multi-layer perceptron (MLP) instead with a sophisticated optimizer (e.g. Adam), which is what we use in our experiments. Since GCNs can take in a feature matrix X as input, in contrast to the heuristic methods detailed above, VGAE can learn node representations using explicit node features, while the heuristics cannot and can only infer structural information about nodes in the graph.

3 Prior Experiments

We first benchmarked the performance of GCNs, Adamic-Adar, and Katz to compare performance on different types of datasets.

3.1 Datasets

We picked two OGB datasets - `ogbl-collab` and `ogbl-ddi`, since they are widely used benchmarks for link prediction tasks. They both have edges that are easy to split into training and test sets by factors like time and target, instead of having to do a random split, so they lend themselves more naturally to the task of link prediction.

- `ogbl-collab` consists of author nodes, with edges between authors indicating a collaboration at a particular time. It also includes feature vectors consisting of averaged word embeddings of the author's papers.
- `ogbl-ddi` is a network of various drugs, with edges between drugs if the drugs interact with each other unexpectedly. We chose this network because it is structurally denser than most social network datasets, and offers a new way to split into train/test - by the protein that a given drug targets.

Details about the datasets are also given in Table 1. For both the datasets, we used the appropriate recommended Hits@K metric to evaluate model performance.

3.2 Methods

To compare the performance of heuristic versus deep learning methods on these datasets, we chose one local heuristic (Adamic/Adar), one global heuristic (Katz), and the GCN method.

For the GCN method, we used a 3-layer deep GCN for the encoding step to generate the node embeddings. We then trained a multi-layer perceptron (MLP) to optimize the aggregation function for the Hadamard product between the node embeddings of pairs of nodes. As previously mentioned, this is a common variation of the structure in the VGAE paper.

Table 1: Information about OGB datasets used

Dataset	# Nodes	# Edges	Features	Train/Test split	Context
ogbl-collab	235,868	1,285,465	Yes	Time	Collaboration network between authors
ogbl-ddi	4,267	1,334,889	No	Protein Target	drug-drug interaction network

Table 2: Hits@K test performance of heuristic and GCN methods

Dataset	Metric	Adamic/Adar	Katz	GCN
ogbl-collab	Hits@50 (%)	64.14 ± 0.0	65.4 ± 0.0	60.5 ± 0.5
ogbl-ddi	Hits@20 (%)	17.3 ± 0.0	11.2 ± 0.0	31.2 ± 2.1

We used mini-batching and the Adam optimizer to train our model, running it for 100 epochs and doing 10 runs to get the average test accuracy and deviation. While training, we sampled positive and "negative" edges (pairs of nodes with no edge between them in the training set) in equal quantities.

3.3 Results

The Hits@K test statistics for the methods are given in Table 2. There is not one method that was consistently the best. Observe that on `ogbl-collab`, Katz and AA gave better results than the GCN, with a statistically significant difference of about 4 percent. However, on `ogbl-ddi`, both local and global heuristics performed significantly worse, with the GCN having the highest Hits@20 percentage of 31.2 ± 2.1. Next, we discuss the potential pitfalls of heuristic and GCN methods and how we may improve consistency using a hybrid approach.

4 Limitations of heuristics and GCNs

From the results above, we can see that model performance varies quite a bit based on the type of dataset. In an extremely dense drug interaction dataset like `ogbl-ddi`, which may not follow the same heuristic assumptions about interactions like a social network does, AA and Katz have very low test accuracies. Here, it is clear that even global heuristics like Katz are not universally applicable, since if all pairs of nodes are separated by a small distance, it is difficult to differentiate any given pair based off of shortest path lengths.

More surprisingly, like in the case of `ogbl-collab`, we see that sometimes simple topology-based heuristic methods can beat the GCN method, which is significantly more computationally expensive, and has the ability to use node feature information. We now suggest instances in which a GCN can fail.

1. GCNs compute single node representations independently, without cognizance of where the target nodes, between which a link existence should be predicted, are located in the network.

   This causes the GCN to lose some topological information. For example, if nodes x and y have structurally similar $k-$hop neighbourhoods, but are far away from each other in the graph, their learned representations will still be similar. Then, for some

given node z, the model's prediction for the existence of an edge $\{x, z\}$ and edge $\{y, z\}$ will be very similar, even if z is only in the neighbourhood of x.

2. Furthermore, the success of the feature smoothing step in graph convolution operations relies on the homophily assumption, that neighbouring nodes have similar features. However, in the case where two nodes have very different feature values, but share many common neighbors, VGAE may not predict a link between these nodes, which might be the incorrect answer in certain types of networks.

It intuitively follows that combining structural information using local and global heuristics with the representations learned by a GCN can help correct the situations in which GCNs fail.

5 Improvements using a hybrid approach

From our initial experiments and analysis, we can identify pitfalls of the baseline GCN method. While more sophisticated deep learning methods for link prediction exist, they are significantly more computationally expensive. Hence, it is natural to ask whether baseline GCN methods can be augmented with simple heuristics to achieve more consistent link prediction performance across datasets, while keeping the computational cost lower.

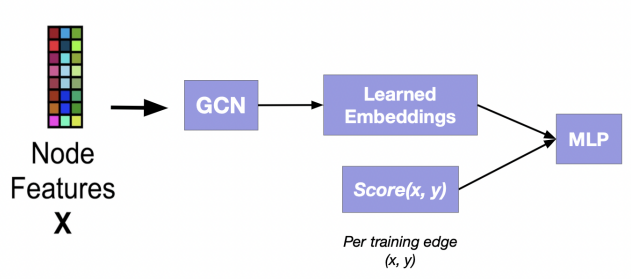
5.1 Comparison to more sophisticated deep learning methods

As we saw, heuristics like AA and Katz make specific assumptions about the underlying network structure, which might not apply to certain types of network structures (for example, in protein interaction networks, 2 proteins having more common neighbours actually implies that they are less likely to interact with each other). There are existing models that build deep learning frameworks with the aim of getting the neural network to learn any arbitrary heuristic rule, to recognize patterns in non-intuitive graphs. One such state-of-the-art method is SEAL [4], which constructs an augmented set of node features before training a GCN.

For any given pair of nodes x and y being trained on (either a positive or a negative edge), SEAL extracts an *enclosing sub-graph* of nodes $h-$ hops away from either x or y, and assigns “node labellings" to every node in this graph, based on the node's shortest path distances to x and y. This information is concatenated with any existing node features, and passed to a GCN. Since this computation is unique for every training pair of nodes, training the model becomes more time consuming, causing an increase proportional to the number of training example, and the number of nodes in the enclosing sub-graph of each training example. The h hop distance of the enclosing subgraph also becomes an additional hyperparameter that needs to be optimized.

In contrast, the augmentations we suggest below are simpler, and do not increase training time and hyperparameter optimization time significantly. We aim to correct some of the known failings of the GCN method, and achieve more consistent (if not exactly optimal) performance across different network structures.

Figure 1: Augmenting learned node embeddings with local heuristic scores



5.2 Introducing local heuristics to learned embeddings

One approach to introducing structural information is to incorporate the pairwise scores computed by heuristics while aggregating single node representations (we do this with an MLP) to predict links. We use the Adamic-Adar index for each pair of node representations, as follows:

Given representations learned for a pair of nodes, x_i, x_j (dimension d), the previous GCN approach passes the element-wise product $\hat{x} = x_i \odot x_j$, as input vector to the MLP (to aggregate single node representations), to determine if we should predict a link between the nodes. Now, we append the Adamic-Adar (AA) index to $\hat{x}$, and pass this $d+1$ dimensional vector into the MLP.

Adding this value augments the learned representations of the nodes with structural neighbourhood information, which also tackles the problem of GCN representations being unable to differentiate nodes at different locations in the graph (**Section 5 (1)**). Since isomorphic nodes at different locations in the graph would have different AA scores w.r.t another node, the embeddings for isomorphic nodes will now differ.

If the graph structures in the training dataset happen to not fit the AA assumption, the MLP can just adapt and optimize the weight matrix to not weigh the AA statistic heavily. More generally, we can pick some local heuristic and append the score $Score(x_i, x_j)$ to the embeddings.

5.3 Augmenting node features with Katz scores w.r.t beacon nodes

Using "beacon nodes" to locate nodes in a network has been shown to be successful for Wireless Sensor Networks [3], and SEAL also constructs a new feature matrix with the aim of adding graph topological information to node features. This motivated another approach to introduce structural information by encoding the "roles" each node plays w.r.t fixed beacon nodes of the network, as features.

We first fix the beacon nodes to be the k most important nodes in a network, determined using an index like PageRank, compute the pairwise Katz score for each node to the k beacon nodes as an $n \times k$ matrix K, and concatenate the feature matrix X with the Katz score matrix K. This method works under the assumption that pairs of nodes that are likely to have an edge between them would have similar Katz scores with respect to the k beacon nodes, and thus their learned representations will be more similar.

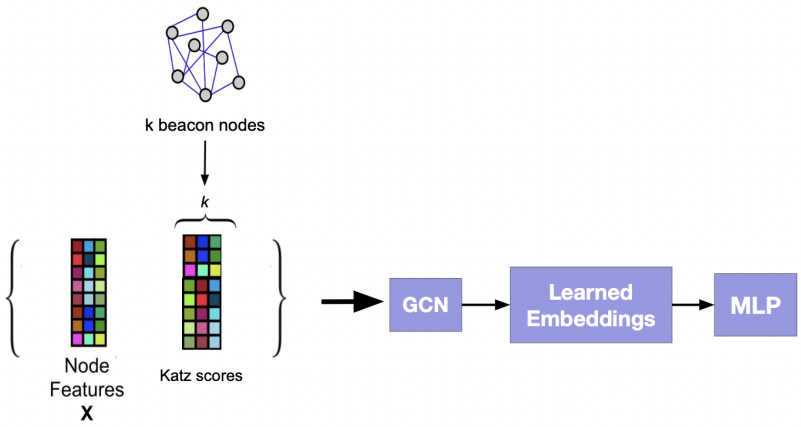


Figure 2: Katz score feature augmentations

Additionally, since it is possible that the top nodes derived through PageRank may be clustered together, we also experiment with a mixture of random nodes and important nodes as beacons, to add more global structural information. We experimented with different values of k, and different random splits for each k, to see if augmenting with a greater number of Katz scores would achieve a significant improvement worth the slight increase in dimensionality.

6 Experiments

6.1 Datasets

We tested our hybrid model on the `ogbl-collab` and `ogbl-ddi` datasets as in the last experiments section. Additionally, we introduced synthetic datasets, using NetworkX's library function to generate stochastic block models. We focused on the case where there are four communities in the graph, and where p, the probability of nodes connecting within a community, is greater than q, the probabilitiy of nodes connecting across communities. This is often referred to as the assortative case for the stochastic block model. Figure 1 on the previous page portrays the general structure of the two-community assortative stochastic block model, which can be generalized to n-communities. For our experiments, we constructed random graphs with a 4-community block model with 40 nodes each, $p = 0.2$, $q = 0.01$.

6.2 Methods

6.2.1 Local Heuristic Augmentation

We first pre-compute the pairwise AA and Common Neighbor scores for each node-pair. The AA scores were computed using the formula in **3.1.1**, and the Common Neighbor's matrix is essentially the square of the network's adjacency matrix. As described before, the GCN learns d length vector embeddings for each node. For each mini-batch of B node embedding pairs, we stack the corresponding B heuristic scores to the $B \times d$ matrix of

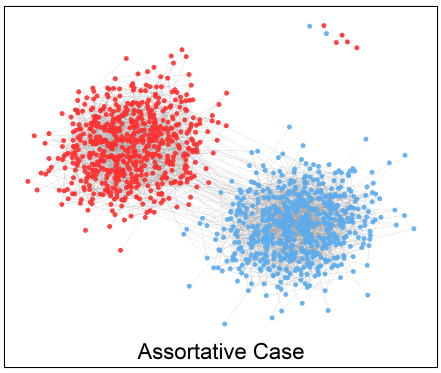


Figure 3: Two community stochastic block model.

Source: Rosey9921, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=103980291

Table 3: Hits@K test performance of heuristics on Random Graphs

Dataset	Metric	Adamic/Adar	Katz
SBM random graph	Hits@10 (%)	24.83 ± 0.0	27.14 ± 0.0
	Hits@20 (%)	46.31 ± 0.0	37.86 ± 0.0

Hadamard products, as described above, and pass the $b \times d + 1$ matrix into the MLP for link prediction.

6.2.2 Beacon Nodes Augmentation

For each network we varied the number of beacon nodes, k, and varied split between important nodes (determined by PageRank index) and random nodes for each k. For `ogbl-collab` we tried k values of 30, 50, 100, 200, and random split %'s of 0, 10, 20, 25, 40 and 50. For `ogbl-ddi` we tried k values of 30 and 50, and the same random split as `ogbl-collab`. For the random graphs, we tried k values 3 and 10, and random split %'s of 0, 50, 100. We use the same GCN - MLP architecture as before. The Katz scores to the beacon nodes were added to each node's feature vector before passing into the GCN. In the case of `ogbl-collab` and the random graphs, since the nodes did not have explicit features, the Katz scores were treated as the sole features.

Additionally, we tested the performance of solely heuristic methods, i.e. Adamic Adar and Katz, on random graphs, as we did on the `OGB` datasets (**4.2**).

6.3 Results

The results for the local heuristic augmentation are in Table 3. Using this method, we get our best Hits@50 percentage of $65.1 \pm 0.6\%$ on `ogbl-collab` using Common Neighbors scores, which is a statistically significant improvement compared to the baseline GCN method (60.5 ± 0.5 %). On `ogbl-ddi`, we get our best Hits@20 percentage of $30.8 \pm 2.0\%$ using AA scores, which is quite similar to the original GCN performance ($31.2 \pm 2.1\%$), and significantly outperforms methods only using heuristics (best heuristic: 17.3 %). On the random graphs, the baseline GCN method outperformed either heuristic augmentation, and solely heuristic methods. However, the augmentations consistently gave us Hits@10

Table 4: Hits@K test performance of Heuristics Augmentation of Embeddings

Dataset	Metric	Baseline GCN	Adamic/Adar	Common Neighbors
ogbl-collab	Hits@50	60.5 ± 0.5	64.40 ± 0.9	65.1 ± 0.6
ogbl-ddi	Hits@20	31.2 ± 2.1	30.8 ± 2.0	29.71 ± 1.0
SBM random graph	Hits@10	52.05 ± 4.51	49.24 ± 4.56	51.56 ± 8.91
	Hits@20	80.43 ± 1.37	75.86 ± 6.56	74.93 ± 2.31

Table 5: Hits@K test performance of Beacon Nodes Augmentation

Dataset	Metric	K (# beacons)	Best random split %	Performance
ogbl-collab	Hits@50	30	10	61.17 ± 0.29
	Hits@50	50	10	61.57 ± 0.38
	Hits@50	100	25	62.34 ± 0.56
	Hits@50	200	20	62.45 ± 0.12
ogbl-ddi	Hits@20	30	10	27.45 ± 1.90
	Hits@20	50	10	29.56 ± 3.50
SBM random graph	Hits@10	3	0	50.00 ± 3.60
	Hits@10	5	0	46.10 ± 4.16
	Hits@10	10	100	57.39 ± 3.45

%'s that were comparable to the baseline GCN across random graphs, while results for solely heuristic methods fluctuated across random graphs.

The results for the beacon nodes augmentation are in Table 4. On `ogbl-collab`, using 200 nodes as beacons, of which 20% are random, we get the best Hits@50 % of 62.45 ± 0.12%, which also outperforms the baseline GCN. On `ogbl-ddi`, 50 beacon nodes, of which 10% are random, give us the best Hits@20 % of 29.56 ± 3.50%, which does not outperform baseline GCN performance. On the random graphs, using 10 beacon nodes, all of which are random, gave us the best Hits@10 % of 57.39 ± 3.45, which outperforms all other methods.

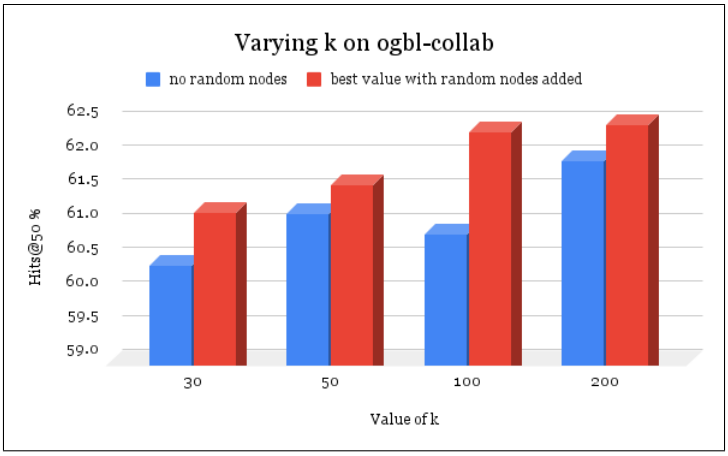


Figure 4: Performance of Katz augmentation on ogbl-collab while varying k.

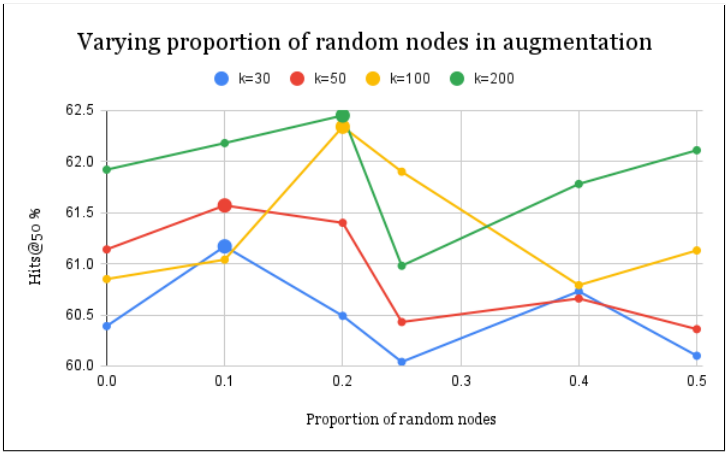


Figure 5: Performance of Katz augmentation on ogbl-collab while varying the proportion of random nodes, for different k.

6.4 Discussion

On `ogbl-collab` both augmentation methods outperformed the baseline GCN, with the augmentation of learned embeddings with heuristics performing the best. This validates our intuition that introducing AA/CN scores to the process of aggregating independently-learned node representations captures structural information that the baseline method does not. On `ogbl-ddi` and the random graphs we saw that heuristic methods just using AA scores performed very poorly (**4.3**), and so it is justifiable that using AA/CN to augment learned embeddings would not significantly improve performance. However, we can see that this method is robust, since even in such a case it performs similar to the baseline GCN method.

The beacon nodes method also outperformed baseline GCN on `ogbl-collab`, which shows that `ogbl-collab` has several important nodes that PageRank was able to identify, that serve as appropriate beacons to infer structural information from.

We saw the intuitive result that the largest number of beacon nodes (200) gave us the best performance, similar to the previously studied result of more beacon nodes improving the localization accuracy of nodes in Wireless Sensor Networks [3], as illustrated in Figure 2. It is possible that using too many beacon nodes would introduce noise regarding the "role" of a node, but `ogbl-collab` was large enough (235,868 nodes) for this to not be an issue. We also saw that performance followed a similar trend w.r.t the degree of randomness in choosing beacons, for different number of beacons, as illustrated in Figure 3. Having just 10%-20% of random nodes as beacons, and the rest being important nodes, resulted in the best performances for each choice of k beacons; larger proportions of random nodes did not give any better performance than no random nodes. This could be explained by the trade-off of using random beacons, where randomness may overcome the issue of important nodes being clustered together, but also introduce noise regarding the network's structural information.

We also saw that the beacon nodes consistently gave us the best performance for random graphs generated using our stochastic block model, among all methods, when we used 10 random nodes as beacons. For future work, it would be interesting to investigate the efficacy

of node localization using beacon nodes on random graphs generated using a SBM, and whether using an index like PageRank to select beacons is at all favorable.

Hence, overall, we see that our hybrid approach gives more consistent results across datasets, compared to just GCN/AA/Katz. Moreover, many other current state-of-the-art GCN methods like SEAL [4] are significantly more computationally expensive, since they involve computing structural labellings for each node based on the edge that we are predicting and also increase the dimensionality of the features significantly. Our methods offer more efficient approaches that incorporate the benefits of GCN architectures as well as heuristic methods.

7 Conclusion

In this paper, we evaluate existing deep learning and heuristic methods for link prediction, and propose two augmentations to GCN models to address the shortcomings of either method. These methods aim to combine structural information using heuristics with independently-learned node representations produced by a GCN. We performed experiments on two datasets, `ogbl-collab` and `ogbl-ddi` - widely used benchmarks for link prediction, and random graphs generated using a stochastic block model. Both methods outperform and match baseline deep learning methods on `ogbl-collab` and `ogbl-ddi` respectively, and significantly outperform solely heuristic methods on `ogbl-ddi` and random graphs.

For future work, it would be interesting to use algorithms other than PageRank to select important nodes as beacons, and use metrics other than Katz to encode the a node's relationship to selected beacons.

References

[1] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[2] David Liben-Nowell and Jon 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.

[3] Guodong Teng, Kougen Zheng, and Wei Dong. Adapting mobile beacon-assisted localization in wireless sensor networks. *Sensors*, 9(4):2760–2779, 2009. doi: 10.3390/s90402760.

[4] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks, 2018.