Temporal Link Prediction on the WikiLinkGraphs Dataset

Nicholas Bowman
Department of Computer Science
Stanford University
nbowman@stanford.edu

Robbie Jones
Department of Computer Science
Stanford University
rmjones@stanford.edu

Semir Shafi
Department of Computer Science
Stanford University
semir@stanford.edu

Abstract

The task of predicting network evolution over time is one that has garnered much research interest, especially as new graph datasets become available in a diverse range of fields. In this paper, we explore methods for temporal link prediction on the new WikiLinkGraphs dataset, which contains snapshots of the Wikipedia hyperlink structure from 2001 to 2018. Broadly, we aim to develop link prediction techniques that can be applied to enhance the structure of the Wikipedia graph, especially for languages that do not benefit from the same number of contributors as the English version. In other words, we want to predict missing hyperlinks that would in the future be added between Wikipedia pages given continued contributions from Wikipedia editors. To do so, we first explore baseline methods that utilize the structural properties of the static WikiLinkGraphs network. Then, as our novel contribution, we frame the challenge as a supervised learning task and apply a number of different Graph Neural Network approaches to the link prediction task. In addition, we apply temporal representation network learning to learn node features that leverage the temporal and evolutionary nature of the dataset we have available to further augment the performance of GNN-based methods. Our experiments show an increase in F1 scores for each of our three GNNs when using temporal node embeddings as inputs over blank inputs. In the end, we conclude that GNN-based methods with time-dynamic input node features provide a promising way forward in accomplishing accurate temporal link prediction for the Wikipedia link graph.

1 Introduction

Link prediction is the task of predicting the existence of unobserved links or future links between pairs of nodes in a network, based on the existing connections in the network and the attributes of nodes and edges (when available). Link prediction is a classic task in the study of graphs, and of great practical relevance to many real-world networks [14]. Successful techniques for link prediction can be used to improve the quality of incomplete graphs, extract missing information, identify spurious interactions, and evaluate network evolving mechanisms. Techniques for link prediction are especially relevant to graphs that have temporal progression, as predictions based on the state of the graph in the past can be used to make predictions about the future structure of the graph.

In this paper, we will consider the interesting task of link prediction in the context of the Wikipedia knowledge graph. Specifically, we will investigate the WikiLinkGraphs dataset published by Consonni

et. al earlier this year [3]. In this graph, the nodes are Wikipedia articles and directed edges are hyperlinks from one article to another. The structure of this graph is constantly evolving, with millions of links being added and removed in the English version of Wikipedia every year. An accurate model of this evolution process could help enhance the editing process for Wikipedia contributors by offering suggestions on what links to add to a certain page to enrich the knowledge graph structure. Additionally, a successful link prediction tool could help automate the process of enriching link structure for Wikipedia in non-English languages that may have a smaller amount of human contributors.

Formally, we frame the challenge of link prediction as follows: Given a sequence of Wikipedia link graphs $G(t) = (V(t), E(t))$ can we predict what new links are added between $G(t_0)$ and $G(t_1)$, where $t_0 < t_1$? There are several key challenges associated with this problem. A good model for link prediction must be able to learn from local interactions between individual nodes while also being cognizant of the overall structure of the graph. This model must additionally be robust to the many outliers that are present in the noisy link structure of Wikipedia. Finally, there is the inherent problem of class imbalance, in the sense that there are relatively few pairs of nodes that actually get connected from time t_0 to time t_1 compared to the number of possible edge pairs among all nodes in the graph.

There is one added wrinkle that distinguishes our problem from previous papers focused on link prediction: our network is not static. Since we are interested in how the network changes over time, and how the addition/removal of edges in the network affects the probability that two nodes will be linked, we develop methods that are sensitive to the time-dynamic nature of the graph. Ultimately, our goal in this project was to develop an approach that would solve all of the unique challenges of doing link prediction on the Wikipedia knowledge graph.

2 Related Work

Given that the WikiLinkGraphs dataset was only released earlier this year, there are no published papers that attempt link prediction on the Wikipedia link graph. However, there has been a significant amount of prior work done on the task of link prediction in general [13]. Below, we detail a selection of both traditional and novel methods for link prediction that have guided our work on this project. Additionally, we discuss work that has been done in the space of adapting traditional graph algorithms to better work on dynamic temporal networks via temporal node embeddings.

2.1 Hierarchical clustering

Clauset et al. present a general technique to infer the hierarchical organization of a graph and demonstrate that this can be leveraged to predict missing links with high accuracy [2]. The authors combined a Monte Carlo sampling algorithm with a maximum-likelihood approach to generate a dendrogram that splits all of the graph vertices into subgroups where each internal node r is added with probability p_r. Each pair of vertices have a likelihood of being connected with probability p_r where r is the lowest common ancestor. Predictions about which links should be added in the future can be made by filtering the pairs of vertices that have the highest likelihood of being connected.

The clustering algorithm can be described with the following steps:

1. Choose a random starting dendogram
2. Run Monte Carlo algorithm until equilibrium.
3. Sample dendograms from the generated chain.
4. For each pair of nodes missing an edge, calculate likelihood of edge existing by averaging the probabilities across the sampled dendograms.
5. Order the pairs by decreasing probability.

2.2 Temporal embeddings

A recent paper from Lee et al. provide a novel way of generating node embeddings for temporal networks, which are networks where the graph edges have a unit of time associated with their appearance [11]. This is important as it addresses one of the open questions proposed by Hamilton et al. in a way that is consistent with foundational advances in node representation learning [7]. Lee

et al. describe the notion of a *temporal walk*, upon which other random-walk approaches to node embeddings can be applied. Simply put, a temporal random walk is a random walk that respects the timeline induced by the edge times. The paper describes ways to bias the temporal walks (much like prior node embedding algorithms) in ways that still adhere to the temporal dynamics of the graph. Finally, the authors show that the embeddings their temporal walks generate perform significantly better on link prediction tasks for temporal graphs compared to node embedding algorithms that ignore temporal information. Figure 1 gives an example of a temporal random walk, whose traversed edges must form a sequence of increasing edge timestamps.

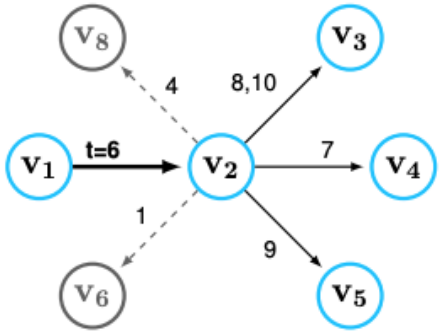


Figure 1: Example of a temporal random walk at time $t = 6$. Note that nodes v_8 and v_6 are not temporally valid neighbors since their edges fall before $t = 6$, so they cannot be explored for the remainder of the walk.

2.3 Graph Neural Networks

Zhang et al. investigate how to learn link prediction heuristics automatically [16]. After examining the local subgraph of the link in question, they aim to learn a function that uses subgraph features and patterns to determine whether a link should exist. They suggest a novel method to learn heuristics from these subgraphs leveraging a graph neural network (GNN). Compared to previous approaches to link prediction, GNNs had a comparatively strong performance. Additionally, Hamilton et. al propose GraphSAGE, which utilizes novel methods for aggregation and updating of node representations in the context of a GNN architecture [6]. Graph Neural Networks form the bulk of our main approach to the link prediction task.

3 Dataset

3.1 Overview

We analyze the WikiLinkGraphs dataset, which has been curated and made available by Cristian Consonni, David Laniado, and Alberto Montresor. As described in [3], the characteristics of the dataset are as follows:

- The dataset comprises of data from 9 different Wikipedia language editions: German (de), English (en), Spanish (es), French (fr), Italian (it), Dutch (nl), Polish (pl), Russian (ru), and Swedish (sv)
- Each language edition of the dataset has an associated table, with the following information contained in each row of the table
  - *page_id_from*: an integer, the identifier of the source article
  - *page_title_from*: a string, the title of the source article
  - *page_id_to*: an integer, the identifier of the target article
  - *page_title_to*: a string, the title of the target article;

  We use an unweighted, directed graph to store this data, with no multi-edges.

- There is a snapshot of the graph for each language provided for each year between 2001 and 2018. This means that there are 18 versions of each knowledge graph available for a given language.
- The links mentioned above have been cleaned to only include links between Wikipedia pages. All external and automatically generated links have been eliminated, leaving an accurate representation of links encoding relationships between certain topics (pages) that have been explicitly added by Wikipedia contributors.
- Tables that contain information about the evolution of the specific language knowledge graphs over time, in terms of the number of nodes and edges over time can be found in [3].

Given that we plan to model the problem of link prediction as a supervised learning task, we must construct a labelled dataset from the baseline graph data discussed above. To this end, we followed the methods described in [8] to build a labelled dataset for each year that we had a snapshot graph for. The labelled dataset construction process is as follows:

- Consider two snapshot graphs $G(t_0) = (V(t_0), E(t_0))$ and $G(t_1) = (V(t_1), E(t_1))$ (for $t_0 < t_1$) in which each directed edge $e = (u, v) \in E(t)$ represents a wikilink from page u to page v and where graph $G(t)$ is a snapshot at year t.
- Generate all pairs of pages (u, v) such that $u, v \in V(t_0), (u, v) \notin E(t_0)$. This represents all possible links between pages that do not currently exist in graph $G(t_0)$.
  - For every pair, associate it with a positive label if and only if $u, v \in V(t_1)$ and $(u, v) \in E(t_1)$. Otherwise, associate the pair with a negative label. This means that the addition of a link in a subsequent year between two pages establishes a positive classification example that we hope to be able to predict.

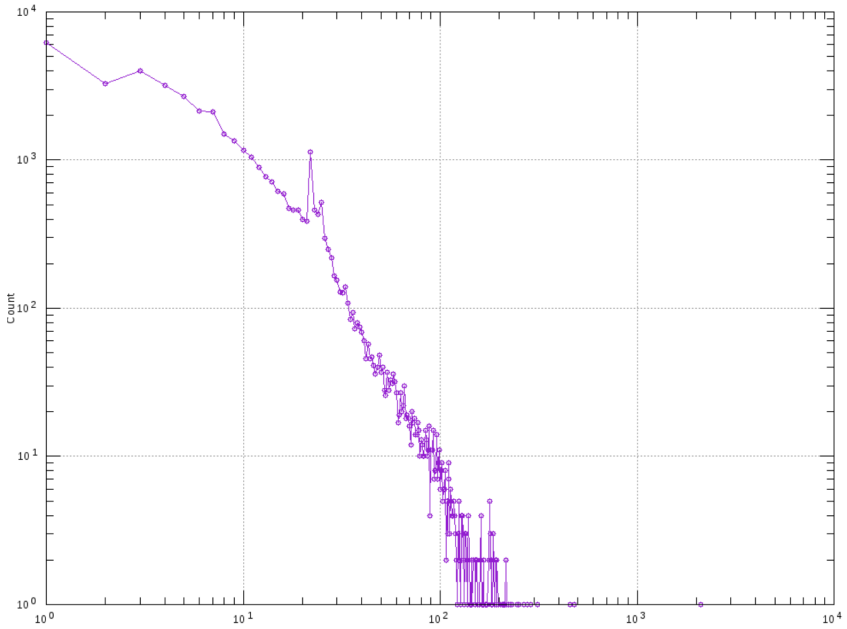


Figure 2: Degree Distribution for 2005 Spanish Wikipedia Graph.

We can see in Figure 2 a right skew where a majority of the nodes have a small out-degree, which leads to an imbalanced classification task (i.e., most of the edges are labelled negative). In order to make sure that we were accurately capturing the class imbalance, that exists in the supervised link prediction task, we generated the dataset to contain 90% negative examples and 10% positive examples. Once this dataset is generated, we follow a standard procedure to segment the dataset into a training set (80%), validation set (10%), and test set (80%). Positive and negative examples were evenly distributed across the training, validation, and test sets.

3.2 Limitations and Final Dataset

One of the major challenges posed by this dataset was its sheer size. Loading the provided edge lists into SNAP to create a single graph could take up to 20 minutes, and some of the larger graphs would crash with fatal out-of-memory errors. Additionally, once we transitioned to training our GNN-based models on Google Cloud GPUs, our maximum graph size was constrained to what could fit on a single NVIDIA T4 GPU (16 GB GDDR6 memory). Due to these numerous hardware limitations, we chose to focus the majority of our experimentation on Wikipedia snapshot graphs from 2006 or earlier.

Specifically, all of the results reported in this paper come from the Spanish 2005 Wikipedia graph ("es-2005"), as this was one of the largest graphs we could computationally afford to use. The evaluation is performed relative to edges that were added to the Spanish Wikipedia graph between 2005 and 2006 (see Section 5.1 for an explanation of our evaluation metrics). The es-2005 graph contains 43,114 nodes and 457,032 edges and there were 194,026 total positive edges that were added to the Spanish Wikipedia graph between 2005 and 2006.

4 Methods

4.1 Methods

We present two different baseline methods: largest common neighbor and hierarchical random graphs. The results of these approaches will be compared against our main contribution, which utilizes temporal node embeddings with graph neural networks.

4.1.1 Baseline Methods

In the common neighbor approach, we examine every pair of nodes and compute the number of shared neighbors those two nodes share. Here, the larger the number of common neighbors, the stronger the argument that those two nodes should have an edge. Since the graph G has a large amount of nodes and because examining every pair of nodes and exploring their neighbors is exponential in complexity, we simplify the problem by only examining pairs of nodes that are 2 hops away (i.e they must have at least one neighbor in common).

Mathematically given nodes X and Y,

$$\text{CommonNeighbor}(X, Y) = N(X) \cup N(Y),$$

where the function N computes the set of neighbors of the node.

Meanwhile, in the second baseline approach, our goal is to find a hierarchical random graph that best explains the observed network. Therefore, we want to maximize the likelihood of generating the observed network from the random model (D, p_r). The likelihood of the hierarchical random graph can be formulated as:

$$L(D, p_r) = \prod_{r \in D} p_r^{E_r} (1 - p_r)^{L_r R_r - E_r}$$

where E_r is the number of edges in G such that the two endpoints of each edge have r as their lowest common ancestor in the generated model, G. L_r and R_r are the number of leaves in the right and left subtree at r respectively.

From here, we sample dendrograms with probability proportional to their likelihood $L(D)$. This Markov Chain converges after approximately $O(n^2)$ steps. Thus, the computational complexity increases with the number of nodes in the graph. To predict which edges are most likely to be added, we sort the results of this process by highest likelihood.

4.1.2 Continuous-Time Dynamic Network Embeddings

Initially, it seemed as if the Continuous-Time Dynamic Network Embeddings (CTDNE) algorithm from Lee et al. might not be appropriate for our problem, given that the WikiLinkGraphs dataset is a

sequence of static snapshots encompassing 18 years, and CTDNE is meant for dynamic graphs with more granular timestamps. Fortunately, we came across the RawWikiLinks dataset, which has exact timestamps for every edge in the WikiLinks data and can be downloaded from the same provider as the WikiLinkGraphs dataset (the WikiLinkGraphs dataset is derived from this RawWikiLinks data). Modeling temporal graphs is still an open problem in network analysis, and temporal node embeddings provide an intriguing tool for solving tasks on these graphs.

As mentioned previously, the CTDNE algorithm relies on a *temporal random walk*, which is a random walk over a graph that respects edge times (i.e., the edges traversed in the random walk must have non-decreasing timestamps). The remainder of the algorithm proceeds akin to other random walk-based embedding generation algorithms, where we find the embedding map that maximizes the likelihood of encountering the other nodes in the temporal random walk (given a certain context window).

4.1.3 Graph Neural Networks for Link Prediction

In applying GNN methods for link prediction, we model the link prediction problem under a supervised learning framework. Based off of the discussion in [8], we can see that a classification model of the link prediction problem needs to predict links by successfully distinguishing the positive examples from the labelled dataset described in the previous section. Thus, the link prediction problem can be posed an a binary classification problem that can be solved by employing effective features in a supervised learning framework. Based off the success of the GraphSAGE model described in [6], we adopt this approach for the task of link prediction on the Wikipedia graph. In order to diversify the set of models that we use, we also choose to explore the use of Graph Convolution Networks (GCN) [9] and Graph Attention Networks (GAT) [15] for the supervised link prediction task. All of these models are implemented using the PyTorch Geometric framework, which allows for easy implementation of methods built on top of the provided MessagePassing module [4].

We experimented with two different input node features for the various GNN methods discussed above. At first, we chose to model having no prior information about the nodes by choose node feature representations as all-ones vectors for all nodes [12]. Then, to incorporate prior information about nodes, we choose the CTDNE node embeddings as the input node features and see if we are able to get better performance than the uniform node representations.

5 Experiments and Results

5.1 Evaluation

Because we are predicting the addition of links over time, we require evaluation metrics that take into account the evolution of our graphs over time and the feasible links that we could expect to predict using temporal algorithms. We define these metrics as follows:

Consider a temporal sequence of graphs $G(t) = (V(t), E(t))$ and two timepoints $t_0 < t_1$. We want to develop a metric that captures how well we are able to predict edges that get added in the temporal evolution of $G(t_0)$ to $G(t_1)$, given that we are only able to observe $G(t_0)$. In order to ignore nodes that get added or removed during the temporal evolution process, we will consider only the induced subgraphs of $G(t_0)$ and $G(t_1)$ on the intersection of the node sets of the two graphs. Formally, let $V_{both} = V(t_0) \cap V(t_1)$ and $I(t_0)$ and $I(t_1)$ be the induced subgraphs of $G(t_0)$ and $G(t_1)$ on V_{both}, respectively. The set of edges that we are interested in, which will act as our gold standard, is the set

$$E_{added} = \{(u, v) | u, v \in V_{both}, (u, v) \notin I(t_0), (u, v) \in I(t_1)\}.$$

In other words, we will positively label edges that were not present at time t_0 but are present at time t_1 among the relevant nodes. Since each of our methods listed below will produce a ranked list of predicted link additions, we can now calculate precision, recall, and F1 score between our predicted sets of edges and E_{added}. These are the final metrics that we will use when reporting our results.

Link Prediction Results on es-2005 Graph

Model	Accuracy	Precision	Recall	F1
Hierarchical Dendrogram	0.031	0.001	0.010	0.001
Common Neighbors	0.241	0.010	0.228	0.019
GCN	0.905	0.520	0.550	0.535
GCN + CTDNE	0.910	0.510	**0.650**	0.552
GraphSAGE	0.920	0.600	0.580	0.590
GraphSAGE + CTDNE	**0.930**	**0.633**	**0.650**	**0.640**
GAT	0.903	0.521	0.204	0.256
GAT + CTDNE	0.885	0.470	0.600	0.520

Table 1: Results of our models' predictions of wikilinks that were added between 2005 and 2006 for the Spanish Wikipedia. Our graph neural networks utilizing temporal node embeddings exhibited the best performance. All three models showed increased F1 score when using CTDNE as input features.

5.2 Implementation and Experiments

For this work, we implemented a complete end-to-end pipeline that reads in the raw data, constructs training and testing datasets for any two consecutive years between 2001 and 2018, extracts features, and trains all of the models described in Section 4. We wrote optimized code using the SNAP C++ libraries to generate Continuous-Time Dynamic Network Embeddings and also re-purposed an existing R codebase to handle generating our hierarchical random graph models. Our implementations can be found on Github [1].

Our GNN architectures all had 3 layers with a hidden dimension of 128. We trained for 300 epochs using an Adam optimizer with a learning rate of 0.01 and a dropout parameter of 0.5. Additionally, for GAT, we used 2 attention heads. To address the class imbalance issue present in our dataset, we chose to weight our supervised learning loss function to more heavily penalize getting positive examples incorrect. For all reported results, the loss for positive examples was weighted three times as much as the loss for negative examples. For our temporal node embeddings, we ran 10 temporal random walks for each node in the graph with a maximum walk length of 80 and context window of 10. The 100-dimensional node embeddings were then optimized with stochastic gradient descent.

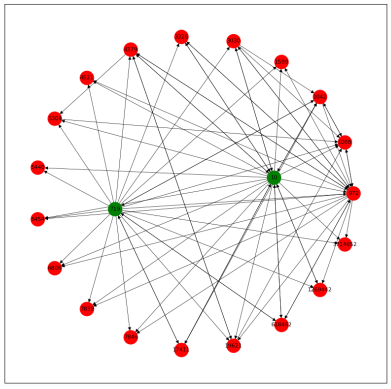
For our baseline methods, we found that it was computationally intractable to run the methods on the entire es-2005 graph, since the baseline methods must consider all possible node pairs, which scales as $O(n^2)$. After approximately 500 to 600 nodes, our computer does not have enough RAM to complete the hierarchical dendrogram algorithm. Therefore, we instead decide to sample 25 newly added edges that we intend to predict and 225 non-existent edges. We run the algorithm on the subset of 500 nodes to then predict the vertices that have the highest probability of an edge. The reported statistics are averaged over multiple trials and subsamplings of the larger es-2005 graph. We followed a similar sampling process when choosing nodes to make predictions on using the common neighbors metric.

As previously discussed, we report accuracy, precision, recall, and F1 score for all methods. Our results are presented in Table 1.

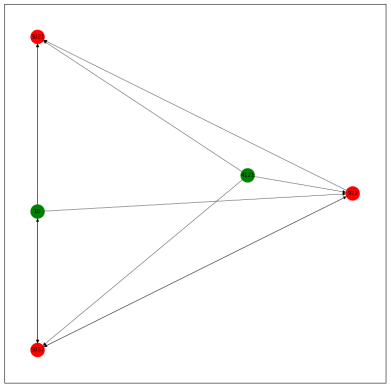
6 Analysis and Discussion

6.1 Overview

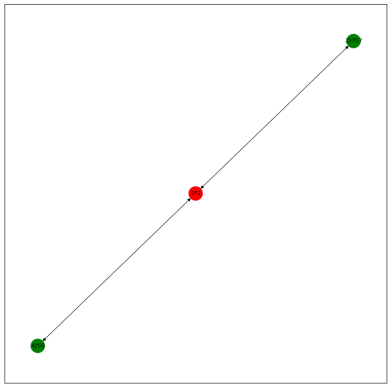
The first clear takeaway from the presented results is that both of our baseline methods both suffered from a lack of scalability, as well as overall poor performance on the temporal link prediction task. While it is possible that we focused on too narrow of a set of baselines, it is clear that traditional proximity-based methods for link prediction are not suited to achieve good performance on graphs as large, complex, and temporally dependent as the Wikipedia knowledge graph. In our experiences running experiments, we found these methods to often be computational-time bound and with very



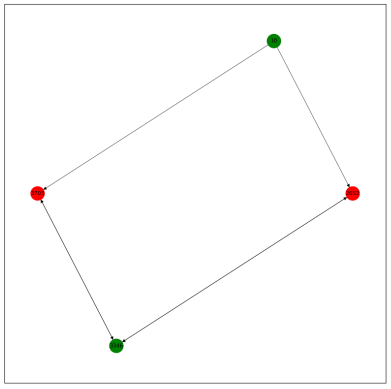
(a) True positive



(b) True Negative



(c) False Positive



(d) False Negative

Figure 3: Examples of 4 different classes of model predictions on the Spanish 2005 Wikipedia Graph by a GraphSAGE model with uniform input node features

limited scope. Therefore, we conclude that GNN-based methods offer the best way forward in terms of developing scalable methods for link prediction on the Wikipedia graph.

The second takeaway is that we saw across-the-board improvement in performance when using CTDNE node features as input into our GNN models, as compared to the baseline uniform node feature input. This gives us reason to believe that temporal node embeddings provide valuable input information for downstream graph tasks, especially when combined with graph neural networks.

Overall, we see that our GAT model struggled to perform as well as the GCN and GraphSAGE models on the supervised link prediction task. In the next section we will discuss our thoughts on why the GAT model struggles, but in this section, we focus our analysis on the GraphSage model, as this model exhibited the best performance. One qualitative analysis that we undertake and that is useful to know is the type of misclassifications that our models frequently make. In particular, we can look at all 4 classes of possible prediction outcomes to gain an insight into our models' learning and predictions. In Figure 1, we present 4 examples of predictions made by our model. In each of the figures, the green nodes represent the two nodes between we make a prediction. In plot (a) of Figure 3 we see an example of a situation in which we correctly predict a node to be added between nodes 10 and 719, presumably as a result of the rich shared neighborhood structure exhibited in the

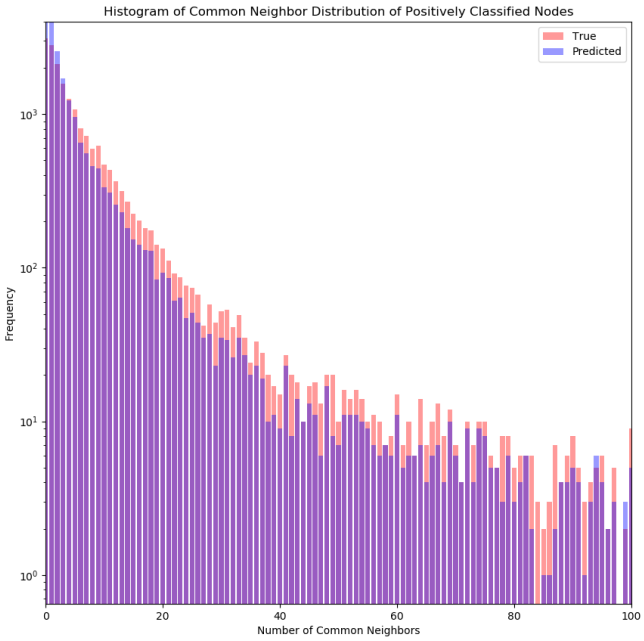


Figure 4: Histogram of node pairs where the buckets are the number of shared neighbors and the frequencies are the number of true edges versus the predicted edges from our GraphSage model.

subplot that we have presented. What is interesting is that we successfully predict the future addition of a link between the pages "Argentina" and "Costa Rica" solely based on network structure, with no higher-level knowledge of the fact that they are geographically co-located countries. In plot (b), we see that our model is able to correctly infer that there should not be a connection between node 10 (Argentina) and node 4121 (Vicente Fox, president of Mexico). As opposed to the prior example, we see that there is a much sparser local shared neighborhood structure for the two nodes, which likely leads to the prediction of this example as negative.

When we start to investigate the false positives and false negatives (plots (c) and (d) of Figure 3), we see that our model understandably starts to lose predictive power when faced with situations where there is little to no shared neighborhood connectivity. In situations like these, our model sometimes predicts negative and sometimes predicts positive, but is generally limited by the amount of information that can reach this certain pair of nodes. In fact, from observations of the common neighbor histogram (Figure 4), we can see that the GraphSage model over-predicts positive examples in situations where there are 0-2 common neighbors, meaning that it is unable to properly fit the true distribution of edges that are added in the future due to signal sparsity. As we will discuss in the further work section, these are scenarios in which we near the limits of what can be accomplished with purely structural methods for the task of link prediction in the Wikipedia knowledge graph. In these cases, having content knowledge fed into the node embeddings would be helpful. However, we are able to show in the end that one can achieve substantial improvement over baseline methods without any actual knowledge of the content of the Wikipedia pages, solely by using GNN-based methods to learn the structure of graph.

6.2 GAT Analysis

One interesting behavior that we noticed during the experimentation process is that our GAT models struggled on the supervised link prediction task. They achieved substantially worse performance than GCN/GraphSAGE and for many combinations of hyperparameters often failed to converge at all after thousands of iterations, often wildly fluctuating in performance until finally reverting back to just

predicting all negative labels. Based off a review of literature, we have come to the conclusion that GAT models generally perform poorly in situations where initial node features are not meaningful [5], which is the case in the experiments where we initialized our node feature vectors to be all ones. However, we do see that when we use CTDNE as input node features, GAT performance stabilizes and starts to approach the success of the other GNN methods.

7 Future Work

There are several interesting directions to pursue for the future of this project. First off, given the ability to use more GPUs and have access to machines capable of loading larger graphs, we would like to investigate the performance of our methods on some of the more recent Wikipedia snapshots. There is a general trend for the Wikipedia link structure to become denser and better developed over time, so it be interesting to evaluate whether our GNN models could adapt to learn the nuances of these denser networks of links. In the absence of more powerful hardware, further research could be done into ways of breaking down message passing calculations to only process sub-portions of the graph at a time, thus helping to make GNN approaches scale better to extremely large graphs. The second interesting direction in which to take this project would be to move beyond the constraint of just using graph structure to predict links added in the future. From the analysis presented earlier in this paper, we may be at the limits of what we can infer purely from graph structure. Therefore, the natural next step is to start including content information about the pages as part of the initial node feature representations, such as doc2vec embeddings [10]. That is, we would look to have node feature representations similar to what is seen in the CORA dataset, where initial feature vectors represent a vocabulary-based indicator of the content of the associated page. By doing so, we should be able to combine semantic information about page content in addition to graph structure to more accurately predict links that get added over time.

Acknowledgments

We want to thank Michele for helping us narrow the scope of our focus in this paper and to gain insight into ways that we could structure the link prediction problem to best achieve success.

Contributions

Nick: Building supervised learning dataset, implementing GNN-based methods, running experiments and doing a hyperparameter search, generating classification visualizations, writing up the report
Robbie: Building data processing pipeline uploading WikiLinksGraph/RawWikiLinks data into Google Cloud (GCS and BigQuery), implementing CTDNE algorithm in C++ and Python, writing up the report
Semir: Implementing common neighbor baseline, hierarchical clustering algorithm, generating RolX embeddings, and helping write paper.

References

[1] `https://github.com/nick-bowman/cs224w-project`.

[2] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98, 2008.

[3] Cristian Consonni, David Laniado, and Alberto Montresor. Wikilinkgraphs: A complete, longitudinal and multi-language dataset of the wikipedia link networks. 2019.

[4] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[5] Liyu Gong and Qiang Cheng. Exploiting edge features in graph neural networks, 2018.

[6] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[8] Mohammad Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. 01 2006.

[9] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[10] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[11] John Boaz Lee, Giang Nguyen, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Temporal network representation learning. 2019.

[12] Jure Leskovec. Graph neural networks. `http://web.stanford.edu/class/cs224w/slides/08-GNN.pdf`, Oct 2019.

[13] Zhepeng (Lionel) Li, Xiao Fang, and Olivia R. Liu Sheng. A survey of link recommendation for social networks: Methods, theoretical foundations, and future research directions. *CoRR*, abs/1511.01868, 2015.

[14] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, Mar 2011.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[16] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.