Predicting Edges in the Iron March Dataset Using Graph Neural Networks

Alex Newhouse
Georgia Institute of Technology
alex.newhouse@gatech.edu

Christopher Roth
croth37@gatech.edu

David Wu
dpwu@gatech.edu

Abstract

The ABSTRACT is to be in fully-justified italicized text, at the top of the left-hand column, below the author and affiliation information. Use the word "Abstract" as the title, in 12-point Times, boldface type, centered relative to the column, initially capitalized. The abstract is to be in 10-point, single-spaced type. Leave two blank lines after the Abstract, then begin the main text. Look at previous CVPR abstracts to get a feel for style and length. The abstract section should contain a brief summary of your work that includes the problem statement, proposed solution and results.

1. Introduction

In 2019, an anonymous account published the entire database of the Iron March far-right extremist web forum on archive.org. This database gave researchers unprecedented access to the internal discussions, organizing strategies, and ideological development of some of the most dangerous terrorist networks active in the West. Since the leak, analysts have dug through the data to investigate extreme posting behaviors and community creation, and law enforcement have leveraged it in cases against actual and would-be terrorists.

With hundreds of thousands of forum posts and direct messages, Iron March also represents a uniquely rich dataset for exploring algorithmic methods for extremism research and counterextremism. As an insular social network, the website's data contains information about how far-right extremists connect with one another and organize for real-world action, both as loose political movements and as discrete terrorist cells. While extensive qualitative work has revealed a trove of significant findings about the social network aspect of Iron March, there has been little research done on the application of automated, predictive methods to the data. Successful creation and deployment of automated relationship prediction methods could significantly increase the capacity of researchers, policymakers, and law enforcement to understand and predict how extreme social networks develop. This would likely prove to be a significant boost in efforts to disrupt and mitigate such social networks.

In this paper, we investigate the use of graph neural networks (GNN) to learn social structures from the Iron March data and apply those learnings to predict the creation of new relationships. We explore the impact of node attributes and different GNN architectures, and we find that GNNs show strong potential for use in the extremism research domain. We also propose several areas where future work can build on these findings.

2. Previous Work

In recent years, GNNs have proven capable of achieving high performance across a number of graph-related tasks, from edge and node classification to edge prediction. In this paper, we focus on the use of deep learning methods for edge prediction, while incorporating some findings from other tasks as well.

Edge (or link) prediction has long been reliant on statistical metrics that may not pick up on many nuances in structure. Early machine learning-based edge prediction work relied on similarity metrics, as described in Liben-Nowell and Kleinberg [6]. In 2017, Schlichtkrull *et al*. [7] pioneered the use of graph convolutional networks (GCN) for the edge prediction task, building on the GCN framework proposed by Duvenaud *et al*. [2] and establishing the practice of autoencoding graph data. With SEAL, Zhang and Chen [8] propose extracting local subgraphs as a heuristic in a GCN-based model, achieving new heights of performance on the edge prediction task.

Gu *et al*. [4] reached state-of-the-art results on link prediction tasks via a modified graph attention network (GAT) called DeepLinker, which demonstrated improvements in computational time, prediction accuracy, and automated discovery of node importance metrics. Further, Izadi *et al*. [5] successfully beat the state-of-the-art score on the CORA node classification task via the inclusion of natural gradient descent in the graph neural network's optimization problem.

Zhang *et al*. [1] describe a method for adding additional knowledge into a graph to assist with link prediction. The authors analyzed a citation graph and used named entity recognition to add additional knowledge edges to the graph. This combats the sparsity problem that is common in most real world graph data. Further extending the idea of mining text for use in supplementing graph data, Zhang *et al*. [1] used a bag of words approach to construct node features.

3. Dataset

The Iron March database is real-world data: it was generated as a byproduct of people interacting with the website and one another, and it was not designed for the purpose of training machine learning models. In order to train the GNN models, we use the essential data in the database: forum posts and direct messages. These represent the thoughts and connections of Iron March's users over the entire course of the website's existence from 2011-2017.

We train and validate the models on the set of forum posts. These posts are "threaded" in a similar structure to Reddit or 4chan posts, which allows for the generation of a social network based on shared interactions within threads. There are over 195,000 forum posts in total, of which roughly 182,000 have an identifiable author and are thus usable. Forum posts consist of a number of features, but for the purposes of this paper we focus on three: the raw text of the post, the author ID, and the thread ID.

In addition, we use Iron March's direct message data in order to further explore the capabilities of the model. There are about 22,300 direct messages, of which roughly 21,000 have an identifiable author. Like forum posts, messages are identified by a thread ID, facilitating the creation of an edgelist.

The full Iron March dataset contains sensitive data, including political beliefs, ages, names, locations, email addresses, and other social account data. The data was leaked without consent of the data subjects, further adding to its sensitivity. However, the dataset has been extensively published and covered in both reputable news sources and academic writing, and it has been used in government documents for the purpose of justifying indictments. Many of its subjects have been widely identified in the media.

We carefully use this data by not including personal information (beyond that which may incidentally arise in forum or message posts) in training the models. In addition, we do not identify any subject in subsequent analysis who has not already been identified in the public record, either through arrest warrants or coverage in reputable sources.

4. Approach

We applied findings from recent edge prediction studies to the novel problem of predicting Iron March edges. Specifically, we explored the effectiveness of graph convolutional neural networks. Although developed primarily for use in analysis of knowledge and citation graphs, we hypothesized that social networks should share a similar enough structure to work effectively in a GNN.

Iron March data does not contain an intrinsic individual-to-individual relationship. There are not friends lists as there are on e.g. Facebook, nor follower/followee relationships as there are on e.g. Twitter. Rather, all relationships are latent within interactions on the forums or in direct messages. As such, we construct an individual-to-individual edgelist via first building a two-mode or bipartite network between individuals and forum threads. From there, we extract individual-to-individual connections if two users post in the same forum thread.

Since this heuristic for generating relationships generally exaggerates the connections–two individuals who happen to post on the same forum thread only once likely are not close friends–we also implement a threshold for inclusion in our edgelist. In this case, we implement a threshold of three: an edge is created between two individuals if they post on three of the same forum threads.

This edgelist is used for training a GNN. Leveraging Pytorch Geometric's built-in dataset splitting functionality, we create an 85%, 10%, 5% split for train, validation, and test data, respectively. We implement the training functionality detailed in Zhang and Chen [8], wherein a number of unconnected node-pairs are selected to create a "negative" edgelist, and concatenated onto the "positive" edgelist. The model can then treat these as the negative and positive labels for the prediction task.

The implementation of this GNN relies on the code in Pytorch Geometric's link prediction example, from Fey and Kim [3]. We customized this code with the functionality necessary to process the Iron March data, generate bag-of-words features based on the text posts, and pass it through the GNN. As part of experimentation, we significantly varied the architecture of the model, ultimately selecting a much deeper and larger model than was used in the original repository. The metrics were modified to output accuracy in addition to AUC and loss, and we included code to generate loss and accuracy curves. Finally, we added functionality to run inference on the entire network after training, so that we can compare the existing network to the predicted network.

4.1. Bag of Words and Other Node Features

The first node feature we used was bag-of-words. This is one method to convert text posts into values for use in the GNN. We first took all the posts and created a vocabulary of all the words. Then, for each author on the forum, we took their posts and counted the frequency of each word they used from the vocabulary, giving us a vector of vocabulary length for each author where the frequency of the i-th vocabulary word used by the author is held at the i-th position. The resulting bag-of-words is quite large, being of shape (number of authors) x (number of words in vocabulary).

To trim down the size of this and make sure the vocabulary contains valid words, the forum posts are pre-processed by removing HTML tags, removing punctuation, and removing stop words (the most common words in a language).

4.2. Model Training & Optimization

The Pytorch Geometric extension was important to training. We based our implementation off of the link prediction example [3] from the Pytorch Geometric documentation. We extended this base model to include more layers and larger layers.

The model takes a list of node features and an edge list as input. The edges are split into train, validation, and test sets. Each epoch the model is presented with all of the training edges. These are fed through a sequence of graph convolutional layers that encode each node. The decoder multiplies the encodings for each pair of nodes to produce an edge probability.

For training, The model produces the probability of an edge existing between each node invovled with a training edge and an equal sized randomly generated set of negative edges that do not actually exist in the network. The network produces probability of an edge between each node in the network. The binary cross entropy loss is used to judge model performance as this is a binary classification task.

We used accuracy to judge the performance of the network. As our results in the section below will show, we did not face problems with overfitting. Our train, validation, and test accuracies were always within 1% of each other.

We used Adam to optimize the network.

5. Experiments and Results

5.1. Model Architectures

Our first attempt to train a GNN on this data utilized the existing, two-layer graph convolutional autoencoding network that is default in the Pytorch Geometric repository. While early experiments showed promise on the smaller direct message dataset, the main forum dataset proved too large for effective training on this simple network. While the loss metric on the training dataset indicated that the model was learning something, the scores on the validation and test datasets revealed that it was not generalizing at all.

As a result, our experimentation largely focused on adding more complexity to the model. The straightforward method of adding more GCN layers, and increasing the parameters at each layer, proved to be effective at learning a model that was able to perform better on the train and test sets. We achieved AUC scores on both sets near 1.0, while accuracy scores ranged from a little over 50% on smaller models to roughly 75% on eight-layer models with 1024 parameters per layer. The randomness of the train/test split in the data resulted in a high variance of accuracy curves, especially in smaller models, so we averaged accuracy scores over 10 runs to compare across architectures.

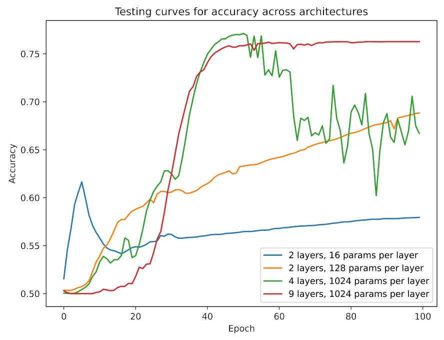

Figure 1. Comparison of accuracy curves for different model architectures.

In order to sanity-check our model, we produced an inference edgelist by passing the entire graph through the best-performing model and comparing it to the original edgelist. The graph for each is shown below.

[Network graphs here]

5.2. Bag-of-Words Effectivenesss

We created a bag-of-words feature vector to represent each user. The hope is that the bag-of-words will represent the kinds of topics the user posts about and help in identifying which users would likley contribute to the same posts. In practice, the bag-of-words did not appear to help the model learn. The ability of the network to predict edges appears to be learned entirely from the edges in the training data.

We trained a six layer graph convolutional network with five different sets of node features. Table 1 shows the best validation and test accuracies for each.

The first row shows the accuracy of the network without the bag-of-words features. In this scenario each node feature is simply a one dimensional tensor set to 1. The next row shows the accuracies when using the 225143 dimensional bag-of-words vectors as features for each user. Accuracy is the same. The bag-of-word vectors are much larger than the number of parameters in the network. The network layers only contain 100s or 1000s of parameters. Also, the counts for each word can be as high as 6000. Scaling the the bag-of-word vectors with the scikit-learn StandardScaler subtracts the mean of each feature and divides by the variance. The result of training with these features is in the third row of the table.

We tried to reduce the dimensionality of the bag-of-words using PCA. Row four of Table 1 shows the results of picking the top 1000 components. Row five shows the results after only keeping the top 50,000 most popular words in the vocabulary. Both of these approaches did not improve accuracy.

The learning rate, random seed, number of epochs, and network layers were all constant for each row in table 1. However, less rigorous experiments with different learning rates and network layer sizes also yielded similar results.

6. Conclusions and Future Work

While the bag-of-words feature vectors did not prove useful for edge prediction, it is optimistic that the network can reach 75% accuracy from the information contained in the edges alone. Better node features would likely improve the accuracy significantly. There are ways of producing smaller embeddings to represent text. Using a language model like BERT to generate embeddings for each forum post and then representing each user by the average of their posts is something we would want to explore next.

We trained the model using our personal computers. This limited the size of the networks we could train. We would expect than a large enough network could simply memorize all of the edges in the training data and achieve 100% training accuracy. We only reached 75% accuracy with our models.

7. Work Division

References

[1] Link prediction with graph neural networks and knowledge extraction, 2020. 1, 2

[2] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 1

[3] Matthias Fey and Dongkwan Kim. Pytorch geometric - link prediction. https://github.com/rusty1s/pytorch_geometric/blob/master/examples/link_pred.py, 2021. 2, 3

[4] Weiwei Gu, Fei Gao, Xiaodan Lou, and Jiang Zhang. Link prediction via graph attention network. *CoRR*, abs/1910.04807, 2019. 1

[5] Mohammad Rasool Izadi, Yihao Fang, Robert Stevenson, and Lizhen Lin. Optimization of graph neural networks with natural gradient descent. *CoRR*, abs/2008.09624, 2020. 1

[6] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007. 1

[7] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017. 1

[8] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 1, 2

Node Features	**Best Validation Accuracy**	**Best Test Accuracy**
no BoW	0.7711	0.7674
BoW	0.7677	0.7639
BoW Scaled	0.7696	0.7654
Bow PCA	0.7725	0.7708
BoW Top 50000	0.7705	0.7659

Table 1. Bag-of-Words Effect on Accuracy

Student Name	Contributed Aspects	Details
Team Member 1	Data Creation and Implementation	Scraped the dataset for this project and trained the CNN of the encoder. Implemented attention mechanism to improve results.
Team Member 2	Implementation and Analysis	Trained the LSTM of the encoder and analyzed the results. Analyzed effect of number of nodes in hidden state. Implemented Convolutional LSTM.

Table 2. Contributions of team members.