Learning from Counterfactual Links for Link Prediction

Aditya Choudhary
21b090001@iitb.ac.in

Sooraj A P
210110104@iitb.ac.in

I. Introduction

As part of our project, we studied the paper 'learning from Counterfactual Links for Link Prediction' [1] and experimented on the code supporting the paper. All our work was done using Google Colab and the notebook we used can be found here.

II. Overview of the Paper

A. The setup

Widely used methods for link prediction utilise node representation learning and use that to learn associations and further predict existence of links. However this approach overlooks the causal relationship between links and nodes. This paper attempts to incorporate the causal relationship between nodes and links in the learning procedure to identify underlying factors that dictate these relations and thereby improve link-prediction models. An example could be, if sat Alice and Adam were both close friends as well as neighbours, a regular model would not be able to look beyond the fact that they are friends because they are neighbours, and would not consider the possibility of them having many common interests of shared friends, etc. The approach of the paper is to ask the counter-factual question '*Would Alice and Adam still be close friends if they were not living in the same neighborhood?*'. This question prompts investigating the causal relations leading to friendship.

Now, in our graph learning setup, we pose the same question as follows: '*Would the link exist or not if the graph structure became different from observation?*'. A counterfactual question is usually framed with three factors: context (as a data point), manipulation (e.g., treatment, intervention, action, strategy), and outcome. Given a pair of nodes and their treatment we look for another pair of nodes that is most similar to our pair but has the opposite treatment. This is a **counter-factual link**.

B. The mathematial Model

Let $\mathbf{A}$ denote the adjacency matrix, $\mathbf{A^{CF}}$ denote adjacency matrix of counter-factual links, $\mathbf{T}$ denote treatment matrix and $\mathbf{T^{CF}_{i,j}} = 1 - \mathbf{T_{i,j}}$. The paper uses the global structural role of each node pair as its treatment. The causal model does not limit the treatment to be structural roles i.e $T_{i,j}$ can be any binary property of node pair (v_i, v_j). The paper proceeds with Louvain an unsupervised approach used for community detection.

The treatment is then taken to be whether or not a pair of node belong to same community or note. Formally,

$$\mathbf{T_{i,j}} = 1 \text{ if } c(v_i) = c(v_j), \text{ and } \mathbf{T_{i,j}} = 0 \text{ otherwise}$$

where c is any clustering method.

Let Z denote the context, T $\in \{0,1\}$ denote treatment and Y denote outcome. The paper uses Individual Treatment Effect (ITE) defined as difference of outcome given context with opposite treatments. Formally,

$$ITE(z) = g(z,1) - g(z,0)$$

$$ITE_{(v_i,v_j)} = g((z_i,z_j),1) - g((z_i,z_j),0)$$

where $z \in Z$ and g(z,T) is the outcome of z given treatment T.

Although the theoretical problem stated above ask for nodes with the same context but different treatment, we settle for those with *similar* context but opposite treatment, as the former, in practice, is unlikely to occur.

That is, we want to find the nearest neighbor with the opposite treatment for each observed node pairs and use the nearest neighbor's outcome as a counterfactual link.

Formally, $\forall (v_i, v_j) \in V \times V$, we define its counterfactual link (v_a, v_b) as

$$(v_a, v_b) = \arg \min_{v_a, v_b \in V} \{d(\widetilde{x_i}, \widetilde{x_a}) + d(\widetilde{x_j}, \widetilde{x_b})\} \tag{1}$$

such that $T_{a,b} = 1 - T_{i,j}$ and $d(\widetilde{x_i}, \widetilde{x_a}) + d(\widetilde{x_j}, \widetilde{x_b}) < 2\gamma$.
Here d(·, ·) is specified as the Euclidean distance on the embedding space of X, and γ is a hyperparameter that defines the maximum distance that two nodes are considered as similar.

C. Learning

Unlike traditional link-prediction algorithms the CFLP model takes as inputs

- The observed graph data $\mathbf{A}$ and raw feature matrix $\mathbf{X}$
- The factual treatments $\mathbf{T}$ and counterfactual treatments $\mathbf{T^{CF}}$
- The counterfactual links data $\mathbf{A^{CF}}$

and outputs the link-prediction logits for factual and counter-factual links.

The model consist of two trainable components: a graph encoder f and a link decoder g. The paper uses GCN as its graph encoder For the link decoder, the paper uses a Multi-Layer Perceptron (MLP) that acts on the context (given by Hadamard Product of node representations of (v_i, v_j)) and the treatment $T_{i,j}$. Formally,

$$\hat{A}_{i,j} = MLP([z_i \odot z_j, T_{i,j}]),$$

	Reported	Calculated (Frobenius)	Calculated (KL)
AUC	0.9212±0.0047	0.9336±0.0044	0.9328±0.0044
AP	0.9392±0.0041	0.9461±0.0038	0.9456±0.0035
Hits@20	0.6809±0.0149	0.6996±0.0278	0.6984±0.0181
Hits@50	0.7701±0.0192	0.7974±0.0149	0.7934±0.0135
Hits@100		0.8490±0.0135	0.8497±0.0120
Best Val		0.9474±0.0030	0.9464±0.0044

TABLE I
KL DIVERGENCE RESULTS ON CITESEER

$$\hat{A}^{CF}_{i,j} = MLP([z_i \odot z_j, T^{CF}_{i,j}]),$$

where [.,.] denotes concatenation.

During the training process, data samples from the factual distribution and the counterfactual distribution are put into decoder g and optimized towards $\mathbf{A}$ and $\mathbf{A^{CF}}$.

The loss function is the standard Binary Cross Entropy loss for both factual and counter-factual distributions given by

$$\mathcal{L}_F = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\cdot log(\hat{A}_{i,j}) + (1-A_{i,j})\cdot log(1-\hat{A}_{i,j})$$

$$\mathcal{L}_{CF} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} A^{CF}_{i,j}\cdot log(\hat{A}^{CF}_{i,j}) + (1-A^{CF}_{i,j})\cdot log(1-\hat{A}^{CF}_{i,j})$$

During the training process, data samples from the factual distribution and the counterfactual distribution are put into decoder g and optimized towards $\mathbf{A}$ and $\mathbf{A^{CF}}$.
The loss function is the standard Binary Cross Entropy loss for both factual and counter-factual distributions given by

$$\mathcal{L}_F = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}\cdot log(\hat{A}_{i,j}) + (1-A_{i,j})\cdot log(1-\hat{A}_{i,j})$$

$$\mathcal{L}_{CF} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} A^{CF}_{i,j}\cdot log(\hat{A}^{CF}_{i,j}) + (1-A^{CF}_{i,j})\cdot log(1-\hat{A}^{CF}_{i,j})$$

The model is trained on both factual and counter-factual distributions however it is tested on just factual distribution. This shift in input distribution has been seen as an issue in counter-factual representation learning. We need the training and test distributions to be similar, hence we force the counter-factual distribution to be similar to the factual distribution, we add regularization parameter to our loss.

$$\mathcal{L}_{disc} = \text{disc}(\hat{P}_f, \hat{P}^{CF}_f)$$

where disc(P,Q) = $\|P-Q\|_F$ where $\|.\|_F$ is Frobenius norm $\hat{P}_{f_{i,j}} = [z_i \odot z_j, T_{i,j}]$ $\quad \hat{P}^{CF}_{f_{i,j}} = [z_i \odot z_j, T^{CF}_{i,j}]$
The overall loss function is given by,

$$\mathcal{L} = \mathcal{L}_F + \alpha\cdot\mathcal{L}_{CF} + \beta\cdot\mathcal{L}_{disc}$$

The discrepancy loss is computed on the representations of node pairs learned by the graph encoder f, so the decoder g is trained with data from both $\hat{P}_F$ and $\hat{P}_{CF}$ without balancing the constraints. Therefore, after the model is sufficiently trained, we freeze the graph encoder f and fine-tune g with only the factual data with respect to $\mathcal{L}_F$.

	Reported	Calculated (Frobenius)	Calculated (KL)
AUC	0.9305±0.0024	0.9320±0.0036	0.9328±0.0031
AP	0.9424±0.0028	0.9440±0.0026	0.9441±0.0022
Hits@20	0.6557±0.0105	0.6383±0.0184	0.6293±0.0217
Hits@50	0.7549±0.0154	0.7537±0.0109	0.7510±0.0144
Hits@100		0.8360±0.0068	0.8364±0.0083
Best Val		0.9386±0.0027	0.9377±0.0035

TABLE II
KL DIVERGENCE RESULTS ON CORA

III. The Datasets

The code was tested on 5 datasets. namely

- Cora
- CiteSeer
- Facebook
- PubMed
- OBG-DDI

The primary metrics used were

- AUC Score
- Hits@20

That being said, both the authors as well as we have tested the code across 5-7 metrics.

IV. Our Proposed Work

Our proposed ideas were ;

1) To test the performance of the code when trained with different norms than the one used in the paper, which was the Frobenius norm.
2) To try and use variables for distances of nodes from communities that were not just 0/1 binaries to try and incorporate the concept of 'closeness from a cluster'.

All our experimental results are from tests run on the CORA and CiteSeer datasets as the others would not load on Colab and we did not have access to a machine running CUDA10-2, which was crucial for us to run the data.

V. Results using KL-Divergence

We experimented by implementing the KL Divergence Loss function in place of $\mathcal{L}_{disc}$ as defined earlier. The results obtained for the same alongside the results we obtained using the prescribed Frobenius Norm and those reported by the authors are tabulated below in tables I for the CiteSeer dataset and II for the CORA dataset. Although there were no noticable improvements, there was a huge reduction in training times. It went from 92 minutes to 77 minutes for CiteSeer and from 62 to 46 minutes for CORA, which is remarkable.

VI. Testing our Multiclass Model

Between each pair of communities we compute a distance metric between them (cosine similarity between centers of the clusters) and linearly assign scores to each distance between 0 and the maximum distance. The treatment of (v_i, v_j) is given to be the distance score between the communities v_i and v_j belong to. Higher the treatment the closer the nodes are.

The counter-factual link (v_a, v_b) would require its treatment to be less than $1 - \mathbf{T_{i,j}}$ i.e

$$(v_a, v_b) = \underset{v_a,v_b\in V}{\arg\min}\{d(\widetilde{x_i},\widetilde{x_a})+d(\widetilde{x_j},\widetilde{x_b}) \mid T_{a,b} \le 1-T_{i,j}, d(\widetilde{x_i},\widetilde{x_a})+d(\widetilde{x_j},\widetilde{x_b}) < \quad (2)$$

The modifications to the authors' code for implementing our multiclass idea can be found here. The results we obtained on running the mentioned code are displayed in tables III and IV

	Reported	**Ordinary**	**Multiclass**
AUC	0.9212±0.0047	0.9336±0.0044	0.9417 ±0.0033
AP	0.9392±0.0041	0.9461±0.0038	0.9546±0.0031
Hits@20	0.6809±0.0149	0.6996±0.0278	0.7542±0.0234
Hits@50	0.7701±0.0192	0.7974±0.0149	0.8291±0.0187
Hits@100		0.8490±0.0135	0.8750±0.0122
Best Val		0.9474±0.0030	0.9583±0.0031

TABLE III
MULTICLASS TREATMENT, CITESEER

	Standard Treatment	**Multiclass treatment**
AUC	0.9320±0.0036	0.9339±0.0039
AP	0.9440±0.0026	0.9443±0.0030
Hits@20	0.6383±0.0184	0.6430±0.0281
Hits@50	0.7537±0.0109	0.7553±0.0166
Hits@100	0.8360±0.0068	0.8338±0.0114
Best Val	0.9386±0.0027	0.9451±0.0033

TABLE IV
MULTICLASS TREATMENT, CORA

respectively.

The most substantial improvement seen was that for Hits@20 on the CiteSeer dataset, where we got an improvement of 5.5% over the results obtained from running the older code. There are also visible improvements in the results for the Hits@50 and Hits@100 values, although not as much as the above mentioned one.
The train time for Multiclass approach was lesser than the standard by a factor of 10% for CiteSeer and 24% for Cora. This was observed because Multiclass converged much faster than standard approach and Early Stopping condition was invoked in a lesser number of epochs

	KCore	**KCore with Multiclass**
Cora	62min.	47 min.
CiteSeer	92min.	83 min.

TABLE V
TRAIN TIMES FOR STANDARD APPROACH VS MULTI-CLASS

REFERENCES

[1] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, Meng Jiang: *Learning from Counterfactual Links for Link Prediction*, ICML 2022 [Attached here].