MTQ Graph Theory

Social Influence Prediction with Deep Learning

Reference paper - DeepInf: Social Influence Prediction with Deep Learning
Jiezhong Qiu , Jian Tang, Hao Ma, Yuxiao Dong , Kuansan Wang, and Jie Tang
Link - https://arxiv.org/pdf/1807.05560

Professor: B.S. Panda
Ayush Chaurasia
Siddhant

Topics

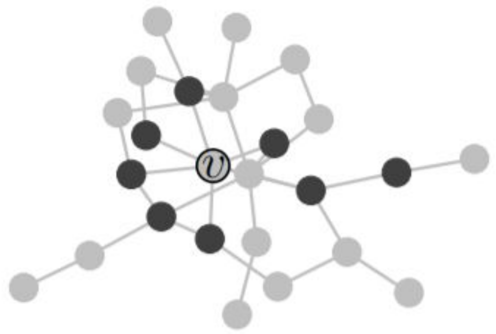
- Problem Statement (Abstract)
- Problem Formulation
- Methodology
- Datasets

Problem Statement (Abstract)

- Social influence typically refers to the phenomenon that a person’s emotions, opinions, or behaviors are affected by others.
- In many online applications such as advertising and recommendation, it is critical to effectively predict the social influence for each individual, i.e., user-level social influence prediction.
- We focus on the prediction of user-level social influence. We aim to predict the action status of a user given the action statuses of her near neighbors and her local structural information

An example of social influence locality prediction. The goal is to predict v’s action status, given 1) the observed action statuses (black and gray circles are used to indicate “active” and “inactive”, respectively) of her near neighbors and 2) the local network she is embedded in.

For the central user v, if some of her friends (black circles) bought a product, will she buy the same product in the future?



Problem Formulation (Few terminologies)

- **r-neighbors and r-ego network:** Let G = (V, E) be a static social network, where V denotes the set of users and $E \subseteq V \times V$ denotes the set of relationships . For a user v, its neighbors are defined as $\Gamma_v^r = \{u : d(u,v) \leq r\}$ where d(u,v) is the shortest path distance (in terms of the number of hops) between u and v in the network G. The r-ego network of user v is the subnetwork induced by Γ_v^r , denoted by G_v^r .
- **Social Action:** Users in social networks perform social actions, such as retweet. At each timestamp t, we observe a binary action status of user u; $s_u^t \in \{0, 1\}$, where $s_u^t = 1$ indicates user u has performed this action before or on the timestamp t, and $s_u^t = 0$ indicates that the user has not performed this action yet. Such an action log can be available from many social networks, e.g., the “retweet” action in Twitter and the citation action in academic social networks.

Problem Formulation

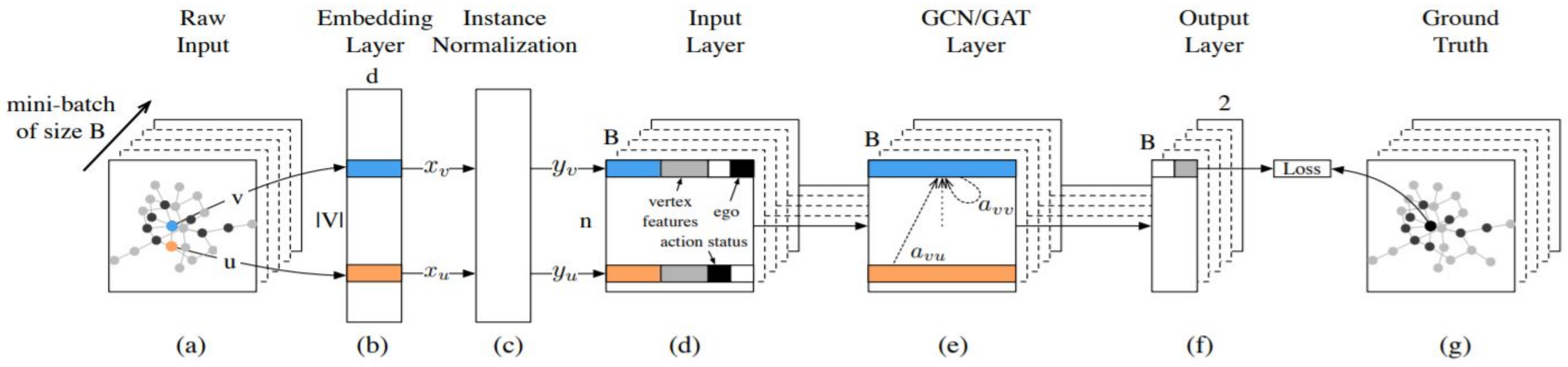
Social influence locality models the probability of v's action status conditioned on her r-ego network G_v^r and the action states of her r-neighbors. More formally, given G_v^r and $s_v^t = \{s_u^t : u \in \Gamma_v^r \setminus \{v\}\}$, social influence locality aims to quantify the activation probability of v after a given time interval Δt:

$$P\left(s_v^{t+\Delta t} \middle| G_v^r, S_v^t\right).$$

Practically, suppose we have N instances, each instance is a 3-tuple (v, a,t), where v is a user, a is a social action and t is a timestamp. For such a 3-tuple (v, a,t), we also know v's r-ego network— G_v^r , the action statuses of v's r-neighbors—s_u^t , and v's future action status at t + Δt, i.e., $s_u^{t+\Delta t}$. We then formulate social influence prediction as a binary graph classification problem which can be solved by minimizing the following negative log likelihood objective w.r.t model parameters Θ:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} \log\left(P_\Theta\left(s_{v_i}^{t_i+\Delta t} \middle| G_{v_i}^r, S_{v_i}^{t_i}\right)\right).$$

Methodology



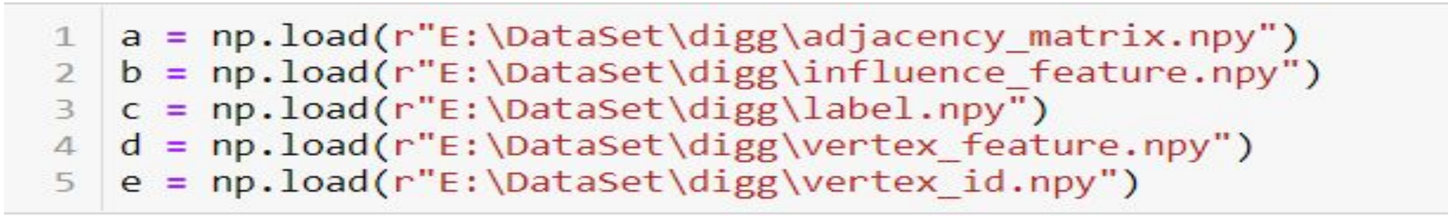
(a) Raw input which consists of a mini-batch of B instances; Each instance is a subnetwork comprised of n users who are sampled using random walk with restart. In this example, we keep our eyes on ego user v (marked as blue) and one of her active neighbor u (marked as orange). **(b)** An embedding layer which maps each user to her D-dimensional representation. **(c)** An Instance Normalization layer. For each instance, this layer normalizes users' embedding xu 's. The output embedding yu 's have zero mean and unit variance within each instance. **(d)** The formal input layer which concatenates together network embedding, two dummy features (one indicates whether the user is active, the other indicates whether the user is the ego), and other customized vertex features. **(e)** A GCN or GAT layer. avv and avu indicate the attention coefficients along self-loop (v,v) and edge (v,u), respectively. **(f)** and **(g)** Compare model output and ground truth, we get the negative log likelihood loss. In this example, ego user v was finally activated (marked as black).

Dataset

- **Digg** is a news aggregator which allows people to vote web content, a.k.a, story, up or down. The dataset contains data about stories promoted to Digg's front page over a period of a month in 2009. For each story, it contains the list of all Digg users who have voted for the story up to the time of data collection and the timestamp of each vote. The voter's friendship links are also retrieved.
- The dataset has node embeddings, feature vectors and adjacency matrix.
- Other datasets were also available like twitter, weibo and oag but we chose to work with digg because other datasets were too large to work with compared to digg.

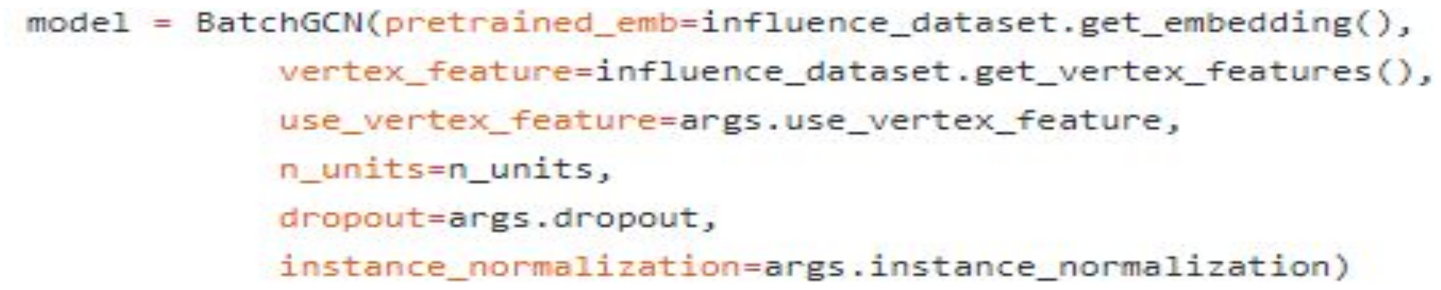
Implementation details

- Implementation done in PyTorch
- Defined a GCN layer
- Stacked GCN layers of different dimensions to form the network
- Used pre trained embeddings given in dataset

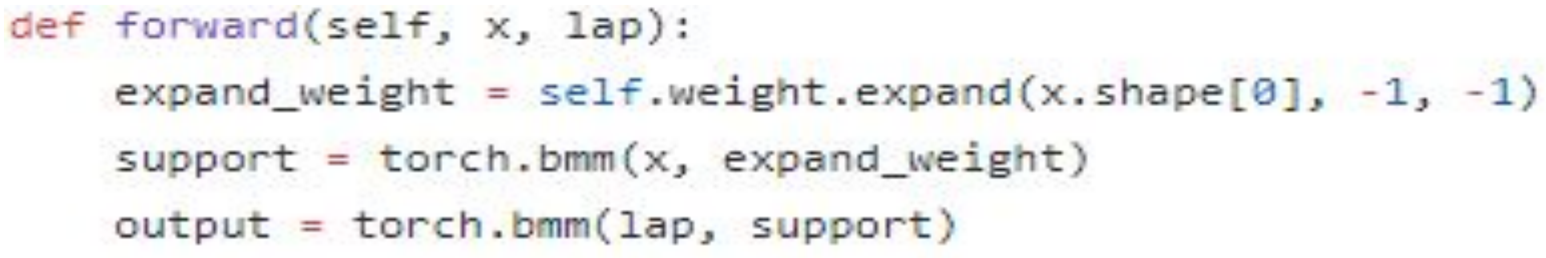


```
a = np.load(r"E:\DataSet\digg\adjacency_matrix.npy")
b = np.load(r"E:\DataSet\digg\influence_feature.npy")
c = np.load(r"E:\DataSet\digg\label.npy")
d = np.load(r"E:\DataSet\digg\vertex_feature.npy")
e = np.load(r"E:\DataSet\digg\vertex_id.npy")
```

Code snippets



```
model = BatchGCN(pretrained_emb=influence_dataset.get_embedding(),
                 vertex_feature=influence_dataset.get_vertex_features(),
                 use_vertex_feature=args.use_vertex_feature,
                 n_units=n_units,
                 dropout=args.dropout,
                 instance_normalization=args.instance_normalization)
```



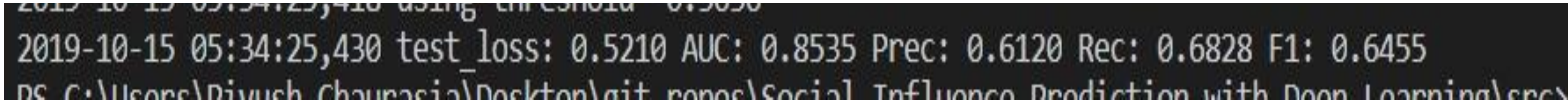
```
def forward(self, x, lap):
    expand_weight = self.weight.expand(x.shape[0], -1, -1)
    support = torch.bmm(x, expand_weight)
    output = torch.bmm(lap, support)
```

Training

- Used 128 x 128 size GCN layers
- 500 epochs
- 32 batch size
- Used Adagrad optimizer

Evaluation

- Evaluation done after 500 epochs
- Metrics used: loss, accuracy, precision, recall and F1 score
- Result obtained is shown below



```
2019-10-15 05:34:25,418 using threshold [illegible]
2019-10-15 05:34:25,430 test_loss: 0.5210 AUC: 0.8535 Prec: 0.6120 Rec: 0.6828 F1: 0.6455
PS C:\Users\Piyush Chaurasia\Desktop\git repos\Social Influence Prediction with Deep Learning\src>
```

Observations and experimentation

- High accuracy but loss on test data is not very low
- Model could be overfitting the data
- Tried using 100 epochs instead of 500 and obtained similar results which implies overfitting
- Tried with less number of parameters in GCN layer, accuracy drops significantly

Further

- Improve embedding of nodes which are given pre-trained here.
- Try combination other variants of GCN like PSCN.

References

- DeepInf: Social Influence Prediction with Deep Learning Jiezhong Qiu , Jian Tang, Hao Ma, Yuxiao Dong , Kuansan Wang, and Jie Tang Link - https://arxiv.org/pdf/1807.05560
- https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-62acf5b143d0
- https://github.com/xptree/DeepInf
- http://keg.cs.tsinghua.edu.cn/jietang/

For all work done by us:
https://github.com/Maestro100/Social-Influence-Prediction-with-Deep-Learning