GRAPH THEORY SEMINAR

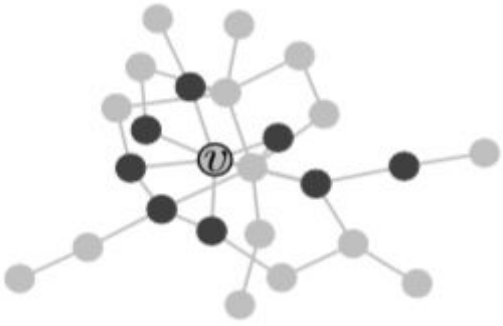
SOCIAL INFLUENCE PREDICTION WITH DEEP LEARNING

AYUSH CHAURASIA (2016MT10617)
SIDDHANT (2016MT10413)

Abstract

Social networking has become an indispensable part of our lives. Facebook, Twitter, Instagram and many other popular social networking sites are used by almost everyone and hence are a very important platform for advertising and expand the customer base. From online shopping to academic courses all are being advertised on these platforms. The important task is how to find an effective social influence of a user so that the user can become a crucial point of advertisement by advertising in his/her social networking circle.

Introduction



In this project, we aim to predict the user-level social influence. We aim to predict the action status of a user given the action statuses of his/her near neighbours or so called the user's in his/her social influence circle. For example, in

the figure aside, for the central user v, if some of her friends(black circles) bought a product, will she buy the same product in the future. We aim to use the modern techniques of machine learning like neural networks, graph convolutional networks, network embedding to make a model for prediction with higher amount of accuracy than traditional methods.

Problem Statement

- **r-neighbors and r-ego network**: Let G = (V, E) be a static social network, where V denotes the set of users and E ⊆ V × 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 retweets. 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 on Twitter and the citation action in academic social networks.

Problem Formulation

Social influence locality models the probability of v’s action status conditioned on herr-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:

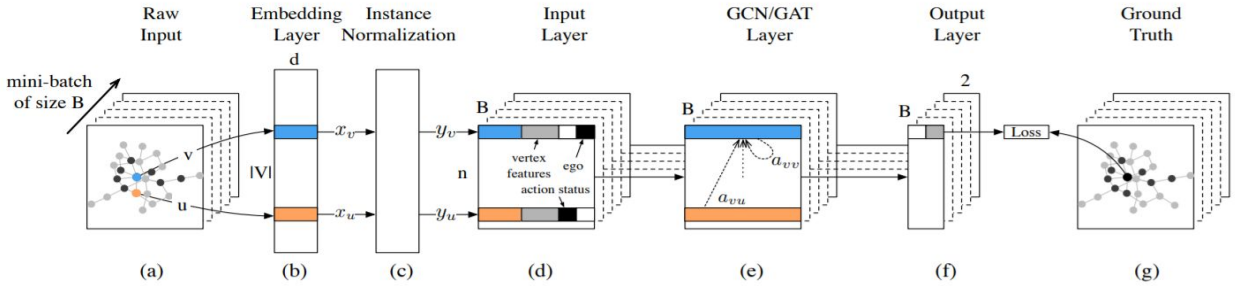
$$P\left(s_v^{t+\Delta t} \middle| G_v^r, S_v^t\right).$$

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 x_u 's. The output embedding y_u '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. a_v^v and a_v^u indicate the attention coefficients along self-loop (v,v) and edge (v,u), respectively.
F. 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).

Datasets

- OAG(Open Academic Graph) dataset is generated by linking two large academic graphs: Microsoft Academic Graph and AMiner. It includes 20 popular conferences from data mining, information retrieval, machine learning, natural language processing, computer vision and database research communities. The social network is defined to be the co-author network, and the social action is defined to be citation behaviors — a researcher cites a paper from the above conferences. We are interested in how one's citation behaviors are influenced by her collaborators.
- 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 Twitter dataset was built after monitoring the spreading processes on Twitter before, during and after the announcement of the discovery of a new particle with the features of the elusive Higgs boson on July 4th,2012. The social network is defined to be the Twitter friendship network, and the social action is defined to be whether a user retweets "Higgs" related tweets.
- Weibo is the most popular Chinese microblogging service. The complete dataset contains the directed following networks and tweets(posting logs) of 1,776,950 users between Sep.28th, 2012 and Oct.29th, 2012. The social action is defined as retweeting behaviors in Weibo—a user forwards(retweets) a post(tweet).
- Dataset Link: https://drive.google.com/file/d/1qBIVdwkKcnOGZnXHcIizzW4_bUekRgC6/view

Graph Convolutional Networks

GCNs are a very powerful neural network architecture for machine learning on graphs.

More formally, a graph convolutional network (GCN) is a neural network that operates on graphs. Given a graph G = (V, E), a GCN takes as input:

- an input feature matrix N × F^0 feature matrix, X, where N is the number of nodes and F^0 is the number of input features for each node, and
- an N × N matrix representation of the graph structure such as the adjacency matrix A of G.

A hidden layer in the GCN can thus be written as $H^i = f(H^{i-1}, A))$ where $H^0 = X$ and f is a propagation. Each layer H^i corresponds to an N × F^i feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer. In this framework, variants of GCN differ only in the choice of propagation rule f.

Hidden Layer

A hidden layer in the GCN can thus be written as $H^i = f(H^{i-1}, A))$ where $H^0 = X$ and f is a propagation. Each layer H^i corresponds to an N × F^i feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer. In this framework, variants of GCN differ only in the choice of propagation rule f.

Propagation Rule Example

$$\mathbf{f(H^i, A) = \sigma(AH^i W^i)}$$

where W^i is the weight matrix for layer i and σ is a non-linear activation function such as the ReLU function. The weight matrix has dimensions $F^i \times F^{i+1}$; in other words the size of the second dimension of the weight matrix determines the number of features at the next layer.

```
In [4]: I = np.matrix(np.eye(A.shape[0]))
        I


Out[4]: matrix([
            [1., 0., 0., 0.],
            [0., 1., 0., 0.],
            [0., 0., 1., 0.],
            [0., 0., 0., 1.]
        ])


In [8]: A_hat = A + I
        A_hat * X
Out[8]: matrix([
            [ 1., -1.],
            [ 6., -6.],
            [ 3., -3.],
            [ 5., -5.]])
```

Problems with the above approach

1. The aggregated representation of a node does not include its own features. The representation is an aggregate of the features of neighbor nodes, so only nodes that has a self-loop will include their own features in the aggregate.
2. Nodes with large degrees will have large values in their feature representation while nodes with small degrees will have small values. This can cause vanishing or exploding gradients, but is also problematic for stochastic gradient descent algorithms which are typically used to train such networks and are sensitive to the scale (or range of values) of each of the input features.

```
In [9]: D = np.array(np.sum(A, axis=0))[0]
        D = np.matrix(np.diag(D))
        D
Out[9]: matrix([
            [1., 0., 0., 0.],
            [0., 2., 0., 0.],
            [0., 0., 2., 0.],
            [0., 0., 0., 1.]
        ])
```

Final Output

- Self-Loops
- Normalizing the Feature Representation
- Activation Function
- Weight matrix

```
In [51]: W = np.matrix([
             [1, -1],
             [-1, 1]
         ])
         relu(D_hat**-1 * A_hat * X * W)
Out[51]: matrix([[1., 0.],
         [4., 0.],
         [2., 0.],
         [5., 0.]])
```

Spectral Graph Convolutions

A recent paper by Kipf and Welling proposes fast approximate spectral graph convolutions using a spectral propagation rule:

$$f(X, A) = \sigma(\mathbf{D}^{-0.5}\hat{\mathbf{A}}\mathbf{D}^{-0.5}\mathbf{X}\mathbf{W})$$

The spectral rule differs only in the choice of aggregate function. Although it is somewhat similar to the mean rule in that it normalizes the aggregate using the degree matrix D raised to a negative power, the normalization is asymmetric.

$$
\begin{aligned}
aggregate(\mathbf{A}, \mathbf{X})_i &= \mathbf{D}^{-0.5}\mathbf{A}_i\mathbf{D}^{-0.5}\mathbf{X} \\
&= \sum_{k=1}^{N} D_{i,k}^{-0.5} \sum_{j=1}^{N} A_{i,j} \sum_{l=1}^{N} D_{j,l}^{-0.5}\mathbf{X}_j \\
&= \sum_{j=1}^{N} D_{i,i}^{-0.5} A_{i,j} D_{j,j}^{-0.5}\mathbf{X}_j \\
&= \sum_{j=1}^{N} \frac{1}{D_{i,i}^{0.5}} A_{i,j} \frac{1}{D_{j,j}^{0.5}}\mathbf{X}_j
\end{aligned}
$$

Training Procedure

1. Perform forward propagation through the GCN.
2. Apply the sigmoid function row-wise on the last layer in the GCN.
3. Compute the cross entropy loss on known node labels.
4. Backpropagate the loss and update the weight matrices W in each layer.

Implementation Details

- Implementation done in PyTorch.
- Defined a GCN layer.
- Stacked different GCN layers of different dimensions to form the network.

- Used pre trained embeddings given in the dataset.

Code snippets

```
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")
```

Above is the embedding file for the Digg dataset.

```
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)
```

Above is the GCN layer made in the code.

```
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)
```

Above is the forward propagation loop.

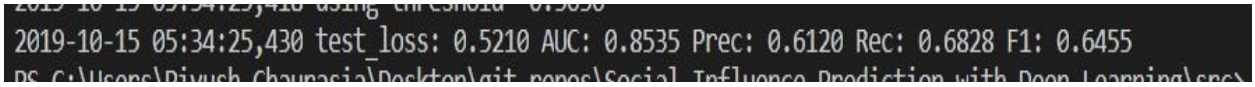
Training

- Used 128 X 128 GCN layers
- 500 epochs
- 32 batch size
- Used adagrad as optimizer

Evaluation

- Evaluation done after 500 epochs

- Metrics used: loss, accuracy, recall and F1 score.



Results obtained are shown above

Observation 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/

Our GitHub repository

https://github.com/Maestro100/Social-Influence-Prediction-with-Deep-Learning