[Experiment] Learning Community Embedding with Community Detection and Node Embedding on Graph (CD)

Anton Begehr
University of Passau
Passau, Germany
a.begehr@fu-berlin.de

ABSTRACT

The graph embedding algorithm ComE developed by Cavallari et al. in their 2017 paper *Learning Community Embedding with Community Detection and Node Embedding on Graph* combines the powers of community detection, community embedding, and node embedding in a closed loop optimizing algorithm to generate low dimensional embeddings for communities and nodes in a graph. This research paper undertakes an experiment with the aims to explore ComE's inner workings and measure its effectivity of classifying crawled twitter data in comparison to Louvain Modularity with normalized mutual information.

1 INTRODUCTION

In their 2017 paper *Learning Community Embedding with Community Detection and Node Embedding on Graph* the authors Cavallari et al. explore graph embeddings by utilizing a three step closed loop optimization process consisting of Community Detection, Community Embedding, and Node Embedding: ComE.[3] They then apply ComE and further graph embedding algorithms on excerpts of multiple, well known graph datasets: Karate Club, BlogCatalog, Flicker, Wikipedia, DBLP. They choose DeepWalk/SF, Line, Node2Vec, GraRep, and M-NMF to measure the quality of ComE's results using Micro-F1 and Macro-F1 for classification results and conductance and normalized mutual information (NMI) for the resulting graph embeddings. It is worthwhile to note, that ComE works with elementary graphs, meaning graphs consisting of nodes and edges, disregarding, for example, node properties, edge properties, and edge weights.

In this paper we will explore the graph embedding algorithm ComE developed by Cavallari et al., generate embeddings using ComE for the Twitter dataset [4] crawled by Rizi, and compare the results to communities gained by applying Louvain Modularity using normalized mutual information (NMI) as a comparison measure.

2 TWITTER DATA

Crawled datasets from Twitter, combined with conference data that is supplied in the public GitHub repository *fatemehsrz/Twitter_Data* will be used.[4]

2.1 Exploration

The supplied twitter data consists of three groups. Each concerning the twitter interactions of a user either attending or not attending one of three conferences. The three conferences are labeled as (1) *anthrocon*, (2) *comiccon2017*, and (3) *icann2016*.

For each conference two CSVs are supplied:

(1) tweets: For each tweet an unique identifier, the content of tweet, an identifier for the user who posted the tweet and a label determining if the suer attended the conference is supplied.
(2) edgelist: Each row represents an edge between two users. It will be assumed, that each edge represents an interaction on twitter. For example commenting and liking would count as interactions.

The tweets id and content will be disregarded in our investigation, because we assume the tweet id does not include information we can utilize since ComE does not support property graphs and we will not be participating in any NLP practices in this experiment.

The tweet and edges datasets of the conferences have the following metrics:

(1) *anthrocon*
  - tweets: 3842
  - users: 1060
  - edges: 15788
  - average edges per user: ≈ 14.9
(2) *comiccon2017*
  - tweets: 4381
  - users: 2495
  - edges: 8203
  - average edges per user: ≈ 3.3
(3) *icann2016*
  - tweets: 2550
  - users: 1016
  - edges: 17045
  - average edges per user: ≈ 16.8

Find the jupyter notebook used for these calculations at *preprocessing/exploratory.ipynb*.

2.2 Requirements

Both the problem space and ComE itself have requirements that the data needs to fulfill. The problem space of graph embeddings is first and foremost classification.

Next to obvious requirements of graph embedding algorithms, the following requirements need to be fulfilled to apply ComE on the provided Twitter data specifically:

(1) The ComE algorithm requires the graph data to be supplied as a **sparse adjacency matrix**. The example code uses a MATLAB .mat file import. The data supplied by Rizi is in an edge list format inside a CSV file. Each row represents an edge from one node to the other.

(2) ComE requires the nodes to be **labeled** for training to determine the number K of clusters it should optimize for, also we will be using the ground truth to test the resulting clusters against.

2.3 Data Preparation

The two requirements of formatting the graph as a sparse adjacency matrix and the nodes with labels that make sense from an application standpoint will be fulfilled with data preprocessing in the existing ComE infrastructure and outside of it utilizing Jupyter Notebooks. These steps are needed to be able to effectively apply the ComE algorithm on the twitter data for community embeddings.

2.3.1 Sparse Adjacency Matrix. ComE takes a sparse adjacency matrix as input for representing the graph on which to apply the algorithm.

The ComE repository includes functionality in *graph_utils.py* for importing the graph as a sparse adjacency matrix from a MatLib file.[2] The graph obtained from crawling twitter data is in the format of an edge list with one edge per row of the csv file. To convert an edge list to a sparse adjacency matrix, two functions were added to *graph_utils.py*:

- *adjacency_matrix_from_edges*()
- *load_csv_edges*()

See section A.3 for details on the implementation.

2.3.2 Labels. ComE requires a labeled training dataset to be able to execute community embeddings. Specifically, ComE determines the number of clusters K from the labels [2] and we will be using the ground truth supplied by the labels to test ComE's resulting communities against.

Jupyter notebooks have been chosen for this task, due to the ability to swiftly iterate on solutions and algorithms, also the size of the supplied data allows for local computation on a personal computer.

The preprocessing procedure for the twitter data mentioned above allows us to utilize ComE's ability to cluster nodes purely from interaction data and then test the results against our ground truth. The ground truth being if a specific user attended an conference or not.

2.4 Labels

The users of each conference dataset correspond to the nodes of the graph. Considering we have three datasets with a user identification column, this raises the following question: Do user identifiers apply across conferences? This question is important to know if the conference datasets can be unioned and create an analysis on the whole dataset instead of doing multiple separate analyses.

If the user identifier applies over all three conferences, the data can be unioned and eight labels (communities) could be extrapolated:

(1) attended no conferences

(2) attended anthrocon
(3) attended comiccon2017
(4) attended icann2016
(5) attended anthrocon, and comiccon2017
(6) attended anthrocon, and icann2016
(7) attended comiccon2017, and icann2016
(8) attended anthrocon, comiccon2017, and icann2016

These eight labels combining the data of all three conferences seemed logical at first and was implemented in the jupyter notebook at *preprocessing/generate_labels-old.ipynb*.

Further data exploration led to the insight, that the user identifiers for all datasets start with the value 0 and all user identifiers between the value 0 and the maximum of the user identifiers for that conference are present in the specific conference. See *preprocessing/user_ids.ipynb* for detailed insights. Both insights suggest, that the user identifier does not after all apply across conference datasets. Therefore, the conference datasets will need to be tested individually. For each dataset there will be a separate binary labeling:

(1) did attended conference
(2) did not attended conference

The implementation for binary label generation can be found here: *preprocessing/generate_labels.ipynb*

3 EXPERIMENT

The experiment was run on the twitter data described in Section 2 using Python 3.6 and PyCharm 2019.1.4 (Professional Edition). All code that has been used and is written to run this experiment is available at the GitHub repository *abegehr/ASDS_ComE*: https://github.com/abegehr/ASDS_ComE.[1]

3.1 Hyperparameters

ComE requires setting multiple hyperparameters. Starting with the number K of communities to be identified, over the size of batches to be used for computation, through to the number of iterations the algorithm should run through, many hyperparameters need to be set. The experiment uses the following hyperparameters:

```
# hyperparameters
number_walks = 10
walk_length = 80
representation_size = 2
num_workers = 10
num_iter = 3
reg_covar = 0.00001
batch_size = 50
window_size = 10
negative = 5
lr = 0.025
alpha_betas = [(0.1, 0.1)]
down_sampling = 0.0
ks = [2]
```

Find following a brief description of each hyperparameter:[3]

- *number_walks*: number of random walks for each node in sampling

- *walk_length*: length of each random walk path in sampling
- *representation_size*: number of dimensions of embedding space
- *num_workers*: number of threads to use for computation
- *num_iter*: number of overall iterations to run
- *reg_covar*: regularization coefficient to ensure positive covariance
- *batch_size*: number of nodes in batch
- *window_size*: windows size used to compute the context embedding
- *negative*: number of negative samples
- *lr*: learning rate
- *alpha_betas*: arrays of combinations of alpha and beta
  - *alpha*: Trade-off parameter for context embedding
  - *beta*: Trade-off parameter for community embedding
- *down_sampling*: relative amount to reduce the sample by
- *ks*: array of number of communities

The hyperparameters have been adjusted to follow advice from the original paper by Cavallari et al. [3]. Further hyperparameter tuning has not been applied.

3.2 Process

The code provided at the GitHub repository *abegehr/ASDS_ComE*[1], which includes the ComE algorithm from the GitHub repository *andompesta/ComE*[2] comes in a state where it is ready to run and results are included in the committed data.

If the input data should be altered to run the experiments on a different datasets, running the experiment requires the following steps:

(1) Generate labels for nodes using the Jupyter notebook *preprocessing/generate_labels.ipynb* and move to *ComE/data/twitter/twitter.lables*.
(2) Copy edge list to *ComE/data/twitter/twitter.csv*
(3) Open *ComE/* in PyCharm. The input and output file paths in *main.py* should be set correctly.
(4) Run using the BICE configuration with Python 3.6.
(5) This will generate multiple files in *data/*. These are the results of the community embedding.

It is also possible to run ComE a different set of graph data. To accomplish this, the edge list and labeled nodes dataset needs to be placed in the correctly named folder inside *ComE/* and the settings and hyperparameters in *ComeE/main.py* should be configured accordingly. See the original paper by Cavallari et al.[3] and the associated GitHub repository *andompesta/ComE*[2] for further details.

3.3 Results

Running the experiment generates multiple files in *ComE/data/*. The following files are generated:

- *g_mixture.joblib*: community embedding in the form of a saved binary of the gaussian mixture model where each gaussian represents one community.
- *labels_pred.txt*: predicted labels for each node in the order or nodes supplied by *twitter.labels*.

- *twitter_alpha-0.1_beta-5_ws-10_neg-5_lr-0.025_icom-62_ind-62_k-2_ds-0.0.txt*: node embeddings for each node where each line holds the node id and one value for each of the *K* dimension of the embedding space.

ComE was executed for all three conferences: (1) *anthrocon*, (2) *comiccon2017*, and (3) *icann2016*. The resulting files can be found at:

(1) anthrocon:
(2) comiccon2017:
(3) icann2016:

3.4 Louvain Modularity

The results from applying Louvain Modularity will be presented and briefly discussed.

4 COMPARISON

The results from applying ComE and Louvain Modularity will be compared utilizing NMI.

5 GRAPH VISUALIZATIONS

The goal is to visualize the results in a two dimensional space.

6 CRITIQUE

Critique of ComE.

For example the number of communities K needs to be supplied for the algorithm to work.

7 ALGORITHM

Deep dive into the technical workings of the algorithm.

For example: Gaussian Mixture Model for community embeddings and likelihood maximization.

8 CONCLUSION

Why is ComE better or worse at this specific task?

REFERENCES

[1] Anton Begehr. [n.d.]. abegehr/ASDS_ComE. data last pushed to GitHub on 2019-09-01, https://github.com/andompesta/ComE.
[2] Sandro Cavallari. [n.d.]. andompesta/ComE. data retrieved from GitHub on 2019-06-17, https://github.com/andompesta/ComE.
[3] Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. 2017. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. ACM, New York, NY, USA, 377–386. https://doi.org/10.1145/3132847.3132925
[4] Fatemeh Salehi Rizi. [n.d.]. fatemehsrz/Twitter_Data. data retrieved from GitHub on 2019-06-17, https://github.com/fatemehsrz/Twitter_Data.

A CODE

A.1 Generate graph

```python
import numpy as np
# union all edge lists into one graph
```

```
# load data
path = "data/graphs/"
edges1 = np.genfromtxt(path+"anthrocon_edgelist.csv", dtype=
    int, delimiter=',')
edges2 = np.genfromtxt(path+"comiccon2017_edgelist.csv", dtype
    =int, delimiter=',')
edges3 = np.genfromtxt(path+"icann2016_edgelist.csv", dtype=
    int, delimiter=',')

# union edges
graph = np.concatenate([edges1, edges2, edges3])
graph.shape

np.savetxt('twitter.csv', graph, delimiter=',', fmt=['%i', '%i'])
```

A.2   Generate Labels

```
import pandas as pd
# generate labels from tweet data

# load data
path = "data/tweets/"
tweets1 = pd.read_csv(path+"anthrocon_tweets.csv", delimiter=',')

tweets2 = pd.read_csv(path+"comiccon2017_tweets.csv",
    delimiter=',')
tweets3 = pd.read_csv(path+"icann2016_tweets.csv", delimiter=',')

def get_attendees(tweets):
    return tweets.loc[tweets['attendance_label'] == 1]['node_id'
        ]

nodes1 = get_attendees(tweets1).to_numpy()
nodes2 = get_attendees(tweets2).to_numpy()
nodes3 = get_attendees(tweets3).to_numpy()

all_nodes = np.concatenate([nodes1, nodes2, nodes3])

node_min = all_nodes.min()
node_max = all_nodes.max()
print("node_min:_", node_min)
print("node_max:_", node_max)

nodes = []
for i in range(node_min, node_max + 1):
    a = (i in nodes1, i in nodes2, i in nodes3)

    if a == (1, 1, 1): label = 7
    elif a == (0, 1, 1): label = 6
    elif a == (1, 0, 1): label = 5
    elif a == (1, 1, 0): label = 4
    elif a == (0, 0, 1): label = 3
    elif a == (0, 1, 0): label = 2
    elif a == (1, 0, 0): label = 1
    else: label = 0
```

```
        nodes.append([i, label])

np.savetxt('twitter.labels', nodes, delimiter='\t', fmt=['%i', '%i'])
```

A.3   Read CSVs

```
import numpy as np
from scipy.sparse import coo_matrix

def adjacency_matrix_from_edges(edges):
    return coo_matrix((np.zeros(len(edges)), (edges[:, 0], edges
        [:, 1])))

def load_csv_edges(file_, undirected=True):
    edges = np.genfromtxt(file_, dtype=int, delimiter=',')

    adj_matrix = adjacency_matrix_from_edges(edges)

    return from_numpy(adj_matrix, undirected)
```