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

Anonymous Author(s)

## 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 deeper and measure its effectivity on classifying crawled twitter data in comparison to Louvain Modularity with normalized mutual information.

**ACM Reference Format:**
Anonymous Author(s). 2019. [Experiment] Learning Community Embedding with Community Detection and Node Embedding on Graph (CD). In *Proceedings of Data Science Seminar*. ACM, New York, NY, USA, 3 pages.

## 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.[2] 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 [3] 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

### 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 encoded as *anthrocon*, *comiccon2017*, and *icann2016*.

For each user, we can extrapolate from the data if the user attended either or multiple of the three conferences. We also have information on the content of the tweet the user dispatched and the tweet id. The last two information points will be less interesting to our investigation, as ComE does not support node properties and we are not participating in any NLP practices in this experiment.

*Data Science Seminar, 2019, Uni Passau*
.

### 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.

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.

ComE also requires the nodes to be labeled for knowing the number K of clusters it should optimize for and testing the resulting clusters against the supplied ground truth.

### 2.3 Data Preparation

These two requirements ComE has on its input data need to be realized by data preprocessing. 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.

Using a Jupyter notebook two datasets were generated for input into ComE:

*2.3.1 Graph.* The graph dataset contains a union of the three edge list datasets, therefore containing interactions concerning all events instead of one of the three.

*2.3.2 Labels.* The label dataset contains labels from 0 to 7 for each node. There are eight labels, because each user can attend none, one, two, or all of the three conferences.

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 which events a specific user attended.

## 3 EXPERIMENT

The experiment was run with the afore mentioned generated datasets graph and labels.

### 3.1 Hyper-parameters

ComE has multiple hyper-parameters.

For simplicity reasons, the hyper-parameters were not tuned. The experiment uses the default hyper-parameters that are present in the ComE GitHub repository for running an experiment on the DBLP dataset[1].

# 4 RESULTS

Running the experiment generates multiple files. One of the files generated is a text-file containing the results of the experiment. The results text-file is named *twitter_alpha-0.1_beta-0.1_ws-10_neg-5_lr-0.025_icom-62_ind-62_k-5_ds-0.0.txt*, containing references to the dataset and hyper-parameters used.

The results text-file has 2495 rows, one for each node. The first row for node 0 reads like follows:

```
0 0.119466 −0.00167963 −0.0100419 −0.34768 −0.11141 −0.194391
      −0.227025 0.265465 −0.256364 −0.144862 −0.201504
      −0.259735 −0.0231989 −0.219537 −0.356864 −0.245884
      −0.0975166 −0.280691 −0.0467756 0.189156 −0.0508059
      0.139592 0.0410849 0.282847 0.0828746 −0.117361
      −0.0487821 0.0498272 0.0363623 0.136442 −0.0659485
      −0.0727705 0.23538 0.0174028 −0.151602 −0.268329
      0.0175012 −0.240817 −0.159485 0.0954141 −0.0240037
      −0.223511 −0.0585588 −0.0422711 0.0720152 −0.0403358
      0.0575148 −0.0742481 −0.260205 0.018126 0.104849
      −0.155449 −0.12444 0.222539 0.102688 −0.271191
      −0.0153477 −0.00814301 −0.0245261 −0.0373094 −0.170265
      0.0142297 0.247277 −0.249435 0.0778326 0.247973 −0.154441
      0.351208 0.0586203 −0.0348056 0.0716461 −0.0177695
      0.164301 −0.193287 −0.0499279 −0.0730012 −0.078206
      −0.0191653 0.130336 0.233464 0.075438 0.0782676 −0.312208
      0.260226 −0.0272545 −0.13308 −0.0962206 −0.294585
      −0.0398387 0.196411 0.0928808 0.00576813 −0.00238099
      −0.0967206 −0.170343 0.13645 0.377029 0.133433 −0.269607
      0.0328925 0.252237 −0.222757 0.136386 0.118541 −0.0201975
      −0.112246 −0.194241 −0.291803 0.102886 −0.0241 −0.244125
       0.146849 0.238158 −0.17695 0.0589843 −0.10204 −0.0934185
      0.0738336 0.0714247 −0.137283 0.380721 −0.34169 0.432387
      0.239321 −0.0428069 0.263721 −0.108774 −0.0988426
```

## 4.1 Louvain Modularity

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

# 5 COMPARISON

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

# 6 GRAPH VISUALIZATIONS

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

# 7 CRITIQUE

Critique of ComE.

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

# 8 ALGORITHM

Deep dive into the technical workings of the algorithm.

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

# 9 CONCLUSION

Why is ComE better or worse at this specific task?

# REFERENCES


[1] Sandro Cavallari. [n.d.]. andompesta/ComE. data retrieved from GitHub on 2019-06-17, https://github.com/andompesta/ComE.

[2] 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

[3] 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

```
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)
```