[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.

ACM Reference Format:
Anton Begehr. 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, 4 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.[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]

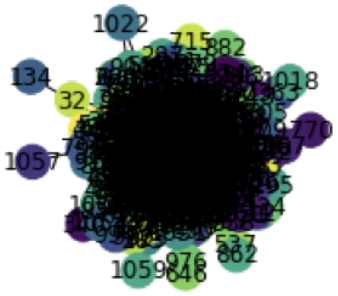


Figure 1: *arthrocon* graph plotted by ComE

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*:

- $adjacency\_matrix\_from\_edges()$
- $load\_csv\_edges()$

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 COME

ComE stands for Community Embedding. A community is a group of nodes that relate to each other through some concept. A community embedding is a vector representation of such a community.

Most solutions in the space of community embeddings do community detection and node embeddings separately and then aggregate the node embeddings for each detected community to create

community embeddings. ComE takes a different, closed-loop, iterable approach. Instead of doing node embeddings and community detection separately, Cavallari et al. realized that community detection can benefit from node embedding, node embedding can benefit from community embedding, and community embedding in turn benefits from community detection.[3] This creates a closed loop that can be iterated on.

ComE defines community embedding as a multivariate Gaussian distribution. The sum of all community embeddings are a Gaussian mixture model. This allows Cavallari et al. to do community detection and embedding together in a single objective function based on GMM.[3] Considering each node embedding to be generated by a multivariate Gaussian distribution from a community allows Cavallari et al. to formulate one likelihood. Given nodes assignments to communities and community embeddings, ComE extends the neural networks of LINE and DeepWalk to preserve first-, second-, and higher-order proximity together and ultimately closes the loop by generating node embeddings through extending the node embedding algorithms LINE and DeepWalk.[3]

4 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]

4.1 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:[3]

```
# hyperparameters
number_walks = 10 # random walks for each node in sampling
walk_length = 80 # length of each random walk path in sampling
representation_size = 2 # dimensions of embedding space
num_workers = 10 # threads to use for computation
num_iter = 3 # overall iterations to run
reg_covar = 0.00001 # regularization coefficient to ensure positive
    covariance
batch_size = 50
window_size = 10 # windows size used to compute the context
    embedding
negative = 5 # number of negative samples
lr = 0.025 # learning rate
alpha_betas = [(0.1, 0.1)] # array of combinations of alpha and
    beta
#alpha: Trade-off parameter for context embedding
#beta: Trade-off parameter for community embedding
down_sampling = 0.0
ks = [2] # 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.

4.2 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.

4.3 Results

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

- *twitter_pre-training.bin*
- *twitter/Twitter.walks.**
- *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.

g_mixture.joblib, *labels_pred.txt*, and *Twitter_alpha-0.1_beta-5_ws-10_neg-5_lr-0.025_icom-62_ind-62_k-2_ds-0.0.txt* can be considered the final results of the algorithm.

ComE was executed for all three conferences: (1) *anthrocon*, (2) *comiccon2017*, and (3) *icann2016*. The resulting files can be found in the GitHub repository *abegehr/ASDS_ComE*[1] at:

1. anthrocon: *results/data/anthrocon/*
2. comiccon2017: *results/data/comiccon2017/*
3. icann2016: *results/data/icann2016/*

To compare the effectiveness of ComE with a proven solutions for community detection on graphs, normalized mutual information (NMI) has been chosen. The normalized mutual information scores for all three conference datasets are as follows:

1. *anthrocon*: $NMI_c \approx 0.0007$
2. *comiccon2017*: $NMI_c \approx 0.0010$
3. *icann2016*: $NMI_c \approx 0.0031$

These are very low values for normalized mutual information. NMI is defined between 0.0 and 1.0, where 1.0 represents perfect, complete labeling of the data. ComE's NMI scores will be compared with NMI scores obtained by applying Louvain Modularity on the same conference datasets.

4.4 Louvain Modularity

Louvain Modularity is an established method for community detection on large networks. It can be used to generate prediction labels from a graph's edges and is less complex in comparison to ComE. To apply Louvain Modularity the number of communities K does not need to be known.

Applying Louvain Modularity on the three conference datasets results in a list of predicted label, one for each node. These labels have been computed in *results/louvain_modularity.ipynb*. Also normalized mutual information scores have been computed for the labels found by applying Louvain Modularity. The NMI scores obtained through the application of Louvain Modularity are listed here:

(1) *anthrocon*: $NMI_l \approx 0.0100$
(2) *comiccon2017*: $NMI_l \approx 0.0181$
(3) *icann2016*: $NMI_l \approx 0.0225$

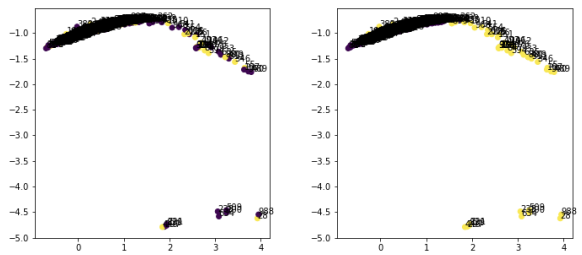
These NMI scores look very similar to the NMI scores obtained by ComE and are very low. Indeed comparable to the NMI scores obtained by ComE.

4.5 Interpretation

Both the normalized mutual information scores obtained by applying ComE (section 4.3) and the NMI scores obtained by applying Louvain Modularity (section 4.4) are very low and represent an insignificant inference of conference attendance from the graph data.

To find out why neither ComE nor Louvain Modularity were able to identify conference attendance from user interactions on Twitter, we can look at one of the conferences in detail. Let's examine the *anthrocon* conference further.

ComE includes a graph utility that provides functionality to plot a graph. The *arthrocon* graph plotted by ComE can be viewed in figure 1. Figure 1 shows a tightly packed graph with most nodes tightly connected and a handful of nodes with low connections.



(a) true **(b) predicted**

Figure 2: *anthrocon* node embeddings colored by labels

Figure 2 shows two scatter plots which display the node embeddings colored with ground truth labels on the left and predicted labels on the right. The structure of the two dimensional embedding does show a similar setup as the graph in that most nodes are highly interconnected and a handful of nodes are separated.

Anthrocon graph and node embedding graphics suggest that the high interconnectivity of the Twitter interaction data provides for increased difficulty of determining the conference attendance. It is possible that the Twitter data does not encompass enough knowledge about the conferences to indicate which users attended the conference and which did not.

The graphs and embeddings resulting from the *comiccon2017* and *icann2016* datasets look similar to the graph and embedding of the *anthrocon* dataset.

5 CONCLUSION

In this research paper, we have run an experimentation on data crawled from Twitter with 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*. ComE's authors claims to combine 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.

The experiment attempted to predict conference attendance from user interactions on twitter. The experiment was run on three datasets: (1) *anthrocon*, (2) *comiccon2017*, and (3) *icann2016*. Normalized Mutual Information (NMI) was used to compare the predicted and ground truth labels and measure ComE's effectivity on the given datasets and problem space. Louvain Modularity was applied as well to compare ComE's results to a proven solution for community detection. Both ComE and Louvain Modularity achieved insignificant NMI scores. A possible reason is the high interconnectivity of users on Twitter, which clouds the knowledge that can be extracted from this data.

Just because this experiment was not able to extract knowledge about conference attendance from the supplied Twitter data, does not mean that the Twitter data does not include this information. Further hyperparameter optimization may lead to better results. Also a down sampling edges may decrease noise sufficiently to provide a clearer view. Increasing processing power and looking at more iterations of applying ComE may also provide further results.

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.