Community Detection on Twitter with Semantically Enriched Data

Gökçe Uludoğan
Department of Computer Engineering
Bogazici University
gokce.uludogan@boun.edu.tr

Işıksu Ekşioğlu
Department of Computer Engineering
Bogazici University
isiksu.eksioglu@boun.edu.tr

İdil Bilge Can
Department of Computer Engineering
Bogazici University
idil.can1@boun.edu.tr

January 16, 2022

1 Introduction

Over the past years, Twitter has become an important platform to discuss various subjects. The website has currently 396.5 million users and in the US, 96% of its users access the platform at least monthly [1].

Tracking and analyzing data on Twitter can lead to important findings such as identifying communities among users. We can define a twitter community as a set of users who share similar opinions and interests. Users in a community have more links within the set than outside of it.

Twitter allows people to voice their opinions and get informed about many events happening all around the world in a fast manner. Events such as the US elections, the murder of George Floyd and many more have been highly discussed in the platform and there has been a significant awareness about them among people, especially US citizens.

Last year's Kyle Rittenhouse trial (he shot three people killing two during the protests last year, charged with homicide and four other counts and later found not guilty) has also caused a reignited discussion about gun rights, violence at protests, what is self defense and what is not on Twitter.

Our project's aim is to detect social communities in terms of interactions and common entity mentions among Twitter users using the tweets about this particular event during a specific period of time.

We collected tweets that are related to Rittenhouse cases as described in next chapter. Then, we determine our ontology and we linked the users and their interests according to the interactions with each other and the entities they use in their tweets. In our study, this linking was done in two different ways and homogenous / heterogenous graphs were created. For community detection, Leiden algorithm experiments were carried out with the homogeneous graph, and in heterogeneous graph, studies were carried out with graph neural network models. The overview of the project can be seen in Figure 1.

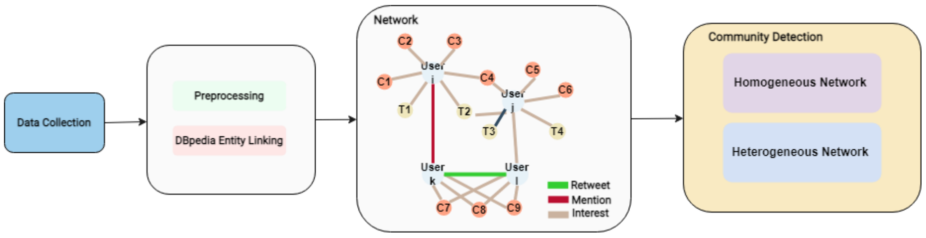

Figure 1: Project Overview

2 Methods

2.1 Data

To analyze the communities, We collect tweets about the trial between September 27, 2021, and December 10, 2021 with keywords "rittenhouse" and "rittenhouse trials". We first use "Tweepy" [2] library to retrieve tweets. However, this library is based on Twitter Search API and retrieves only tweets from the previous week. Thus, we collect the rest of the tweets via "Scweet" [3], a library scraping Twitter. Once collection is completed, we clean the tweets with "Tweet preprocessor" [4] and extract mention and retweet features. These features are widely used in modeling users and communities since these interactions might indicate similarity of user views. The data statistics are given in Table 2.1.

Table 1: Data statistics

Number of Tweets	231711
Number of Unique Users	141862
Average Number of Tweets per User	1.63
Average Number of Mentions per User	2.21
Average Number of Retweets per User	1.63

Interactions between users are key elements while modeling networks. Yet, the number of interactions can be limited. In such cases, the tweet content is used to remedy this limitation. Since the interactions between users are sparse in our dataset (see Table 2.1), we enrich data by utilizing content. Specifically, we annotate tweets with DBpedia Spotlight to extract entities that users are interested in [5, 6]. We speculate that the users belonging to same communities are more likely to have common interests (i.e. entity mentions). Thus, we use entity/interest similarity between users to model user similarity along with retweets and mentions.

There are, however, some limitations of entity linking. We inspect the annotation results and see that either DBpedia Spotlight or the entity linker built on top of it does not produce highly accurate results. The results can be further filtered. However, we omit this step since we have limited time. In addition, we realize that there are some entities that are not linked to DBpedia although there exists resource describing them. Since most of these entities are phrases or concepts that might be useful for this context, we also include them.

2.2 Ontology

We develop an ontology to model user interest in entities and interactions. All classes, properties and overview of the ontology are shown in Figures 2 and 3

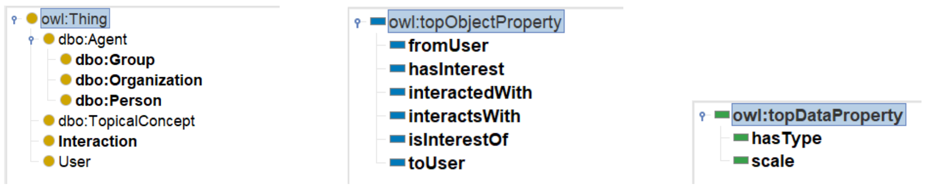

Figure 2: Our Ontology - Classes, Object properties, Data properties

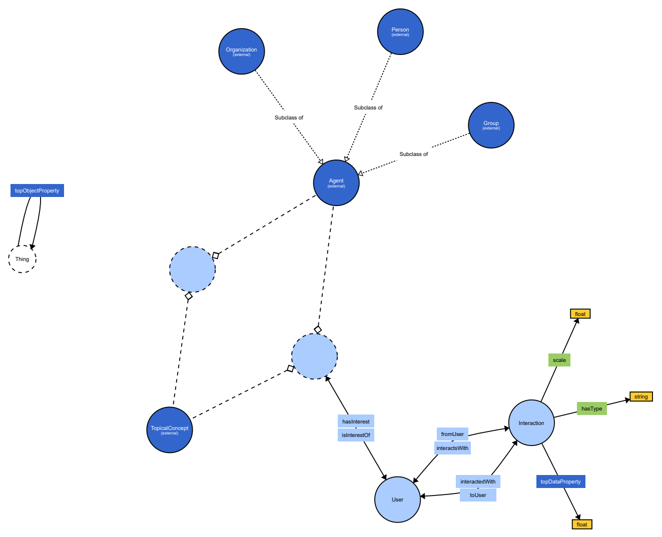

Figure 3: Ontology Overview

The ontology consists of a set of classes representing interaction (i.e. mention or retweet), user and entities. “User” is the class corresponding to a user while “Interaction” is the one representing an interaction between two users. This class has two data properties which are “hasType” indicating whether the interaction is a “mention” or “retweet”, and “scale“ storing the strength of the interaction. Furthermore, this class is “domain” of two object properties: “fromUser” and “toUser”, which denotes the user initiating the interaction (interactsWith) and the one being interacted (interactedWith), respectively.

The classes except “User” and “Interaction” represent the DBpedia entity classes the users frequently mention in their tweets. After annotating the tweets, we take a look at the frequency of the entities and observe that the tweets contain lots of entities belonging to “dbo:Agent” class and decide to include this entities of this class and the most frequent subclass: “dbo:Person”, “dbo:Group” and “dbo:Organization”. In addition, the ontology includes a class named “dbo:TopicalConcept” for the aforementioned concepts/phrases that cannot be linked to DBpedia with the entity linker we employed. User-interest (or entity) relationships are described with object properties named “hasInterest” where the domain is “User” and the ranges are “dbo:Agent” and “dbo:TopicalConcept”; “isInterestOf” where domains are “dbo:TopicalConcept” and “dbo:Agent” and the range is “User”.

2.3 Model

Following the ontology, we build a graph by populating extracted users, mentions, retweets and interest (entities) with “rdflib” [7]. Then, we use GraphDB [8] to store and query the graph. Figure 4 shows a set of relationships for a user and for an entity/interest. The user on the left figure “hasInterest” “dbo:White_supremacy” and “dbo:Fascism” and also a set of interactions shown in purple. On the other hand, the entity/interest at the center of the right figure “isInterestOf” a set of users shown in blue and is a “dbo:TopicalConcept”.

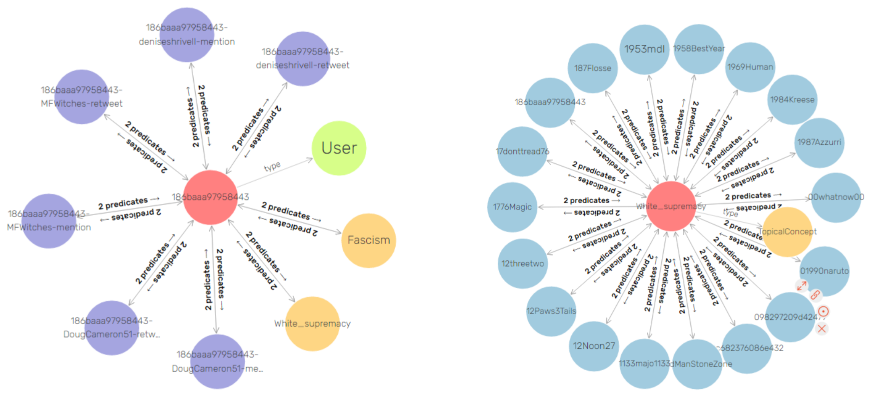

Figure 4: GraphDB Examples

2.3.1 Homogeneous Model

Homogeneous model is a weighted directed graph where nodes represent users and edges represent similarity between users. This graph is constructed by calculating similarity between any two users. Let u_1 and u_2 is a user pair, their similarity can be computed with the following formula:

$$w_{u_1,u_2} = (\alpha_M * w_M(u_1,u_2) + \alpha_{RT} * w_{RT}(u_1,u_2) + \alpha_I * w_I(u_1,u_2))$$

where w_M, w_{RT} and w_I corresponds to mention, retweet and interest similarity scores while α_M, α_{RT}, α_I are the corresponding parameters weighting the importance of these scores. w_M and w_{RT} scores are computed by counting how many times a user mentions or retweets other users and normalizing the counts. For instance, if user A mentions user B twice and user C once, then $w_{A,B}$ becomes 0.66 and $w_{A,C}$ becomes 0.33. On the other hand, we adopt a class based approach. For each interest class with no subclass (i.e. “dbo:Person”, “dbo:Group”, “dbo:Organization”, and “dbo:TopicalConcept), we compute the cosine similarity between user vectors for the corresponding class and average the scores to obtain w_I. At each class level, a user is represented with a vector containing the inverse document frequency for the entities this user is interested in and 0 for the other entities.

Once the graph is constructed, we apply a community detection algorithm on this graph using "cdlib" [9]. The algorithm is named as "Rb_pots" in this library and equivalent to Leiden algorithm with specific parameters we use. Leiden algorithm is a popular and efficient algorithm for community detection. It contains three steps which are local moving of nodes, refinement and aggregation of nodes. First, nodes are moved locally to maximize modularity. Second, the nodes within the communities proposed by the previous step are refined and merged if necessary. Then, the nodes are aggregated according to the refined communities.

To evaluate the resulting model, we use two metrics: surprise and Newman Girman modularity. Surprise is a quality metric based on statistics and the assumption that edges between nodes emerge at random according to a hypergeometric distribution. The higher surprise score means that the partition is less likely to be resulted from random generation. Hence, the higher the surprise, the better the community structure. The second metric we report is Newman-Girman modularity which compares the fraction of intracommunity edges with the expected number of such edges according to a null model.

2.3.2 Heterogeneous Model

Heterogeneous graphs contain different types of nodes and edges. As it can be understood from the ontology we created, the interest and users are different types of nodes. For this reason, the methods developed for heterogenous graphs can be easily applied to the populated ontology.

In this part of our study, user-interestType-interest triplets were extracted using SPARQL and a heterogeneous graph was produced. The input created in this way is given to the CP-GNN method.

Context Path-based Graph Neural Network (CP-GNN) is a graph neural network created for heterogeneous networks. It can deal with high order relations and by using an attention mechanism, it can identify important types of nodes when creating node embeddings. As in all graph neural networks, this method tries to maximize its objective function by using the embeddings it creates. In CP-GNN node embeddings are extracted according to context neighbors by using its three components. The first component is embedding transformation. This component maps the learned representation to the primary graph. Sişnce our main target is the primary graph, this component is very useful for reducing parameters. The second component is the context path graph NN layer. This component calculates and aggregates attention scores. The last component is an objective function that tries to maximize the probability of having the context neighbors for each node in the primary graph.

3 Results

3.1 Homogeneous Model

For community detection, we constructed a set of homogeneous models with varying α values. While all retweet and mention relationships were included in the graph, this was not the case for the interest relationships. Given that there were around 140K users in the dataset, computing and storing the similarity of each pair was computationally heavy in terms of both time and space. For this reason, we only included a subset of them. Then, we executed the community detection algorithm on each graph obtained and reported the results in Table 2.

Table 2: Community Detection Results for Homogeneous Model

α_M	α_{RT}	α_I	**surprise**	**newman girvan**
1	-1	0	4.15×10^5	0.92
0	1	0	2.36×10^5	0.87
1	0	0	5.62×10^5	0.87
1	1	0	5.62×10^5	0.85
1	1	1	1.51×10^6	0.83
1	0	1	1.50×10^6	0.82
0	1	1	1.19×10^6	0.81
1	-1	1	1.42×10^6	0.81
0	0	1	2.81×10^6	0.79
1	-1	-1	1.43×10^6	0.78
0	1	-1	1.19×10^6	0.76
1	0	-1	1.49×10^6	0.74
1	1	-1	1.50×10^6	0.69

The results show that the highest modularity score is obtained with the model using positive mention weights and negative retweets weights. This might imply that there exists a negative correlation between these features in this context. Additionally, we observe that adding interest relationships with positive weight increase the surprise score while slightly decreasing modularity scores. However, these results are preliminary and require further validation.

We also visualized two of the resulting community partitions with Gephi [10]. Figure 5 shows the communities detected with the graph using only mentions on the left and the one using mention + retweet + interests on the right. While the communities on the left are smaller, the communities on the right contains two large communities (green and purple ones) compared to others. We randomly sample a few users and determine their stance about the trial by investigating their tweets. These examples suggest that the users in the green community are more likely to be against Kyle Rittenhouse. However, further analysis is necessary.

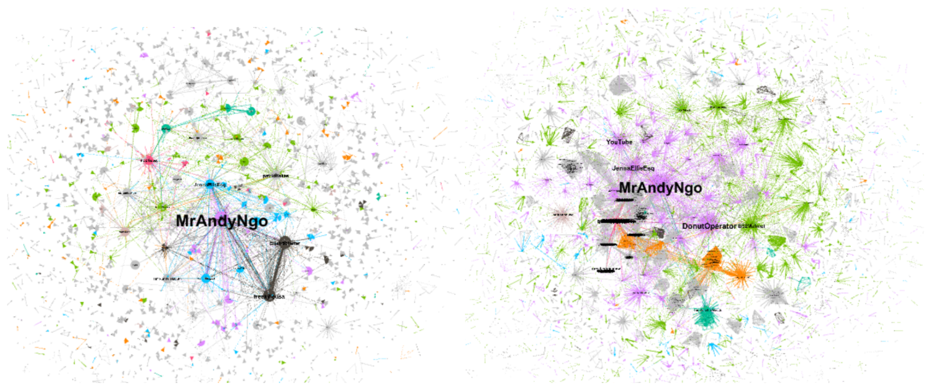

Figure 5: Community Visualizations

We also developed an application showing the words with highest TF-IDF scores in each community.

3.2 Heterogenous Model

In the experiments conducted at the original paper of CP-GNN, target networks were tried to be estimated by using input graphs and two networks have different types of nodes. As can be understood from these experiments to use CP-GNN, nodes should have labels. In our case, users are input nodes and entities are target nodes. So by using a user network, we will try to predict the right entities for each user.

Node classification result of this experimental can be summarized as macro f1 score is 0.72 and micro f1 score is 0.2.

There is no label of any node in the graph created with our ontology. We have extracted the sub-network that can be given as input to CP-GNN, but the targets in this network are very diverse. There are 256 different entities (targets) in total. In other words, the input graph is not suitable for the classification task. For this reason, the classification results are very low.

4 Discussion

In experiments with homogeneous graphs, after arranging the input data, the Leiden algorithm seems to be able to distinguish communities with different views for the Rittenhouse case.

Since the GNN method we use makes node classification, it is not compatible with the graph we created. There is no label of any node in the graph so we accept entity nodes as targets but the number of different entities is 256. For this reason, the classification report is nor successful. Also, because of the limited time and limited computational power, we couldn't add retweet and mention interactions to the input results. Maybe, as in homogenous graph, adding these relations can improve the classification results.

5 Future Work

We plan to focus on heterogeneous graphs in our future studies. Since we do not have proper node labels, we could not fully benefit from the GNN method. Unsupervised methods can be used to correct this situation.

In addition, we could collect tweets according to certain sentiments, or we could automatically label them using a sentiment classification method. Another automatic labeling method can be done based on the interaction with the user who has a certain view for Rittenhouse case.

6 Conclusion

In this study, tweets related to the Rittenhouse case were collected and community detection was attempted by using certain interactions of these gathered users. For this purpose, an ontology was developed and two different types of graphs, homogeneous and heterogeneous, were obtained.

According to results that were obtained in experiments with homogeneous graphs, Leiden algorithm seems to have identified users with different opinions and when we analyse the results further, it was observed that each interaction information we included contained useful information for community detection. We could not benefit from heterogeneous graphs as desired. The reason for this is that we do not have an appropriate input for the graph neural network method we use. In future studies, we plan to obtain successful results for heterogeneous groups by collecting tweets according to specific labels or by using automatic labeling methods.

References

[1] https://backlinko.com/twitter-users.

[2] Joshua Roesslein. Tweepy: Twitter for python! *URL: https://github.com/tweepy/tweepy*, 2020.

[3] https://github.com/altimis/scweet.

[4] https://github.com/s/preprocessor.

[5] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python. *Zenodo*, 2020.

[6] https://github.com/martinomensio/spacy-dbpedia-spotlight.

[7] Carl Boettiger. rdflib: A high level wrapper around the redland package for common rdf applications, 2018.

[8] https://www.ontotext.com/products/graphdb/.

[9] Giulio Rossetti, Letizia Milli, and Rémy Cazabet. Cdlib: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science*, 4(1):1–26, 2019.

[10] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*, 2009.