Kmeans Based Community Finding

Gaurav Singh

Abstract. There are a number of community finding algorithms present like **fastgreedy**, **edge betweeness**, **walktrap**, **label propogation** and **infomap**. All of these algorithms are fairly complex, some of them in terms of run time complexity and other in terms of development complexity. I decided to develop a simple approach to community detection in graphs using a kmeans based clustering.

1 Algorithm Description

1st Approach
I load the graph in the memory using networkx. After that I represent each node in the graph as a vector representation of all of its neghbours. It basically means, I denote each node with huge vector V of size N (N is total number of nodes in the graph) such that there is 1 in i_{th} position if ith node is a neighbor of given node, else the value at ith position is 0. Thereafter, I cluster the nodes using k-means clustering algorithm. All the nodes that are in a given cluster constitute one community.

2nd Approach
After analyzing the graph given in Gephi and using matplotlib, I realized that some nodes which have really high degree, can have good similarity with a lot of other nodes, therefore in order to compensate for this effect, I replace the value at i_{th} bit from 1 to $\frac{1}{number\ of\ neighbors(neighbour_i)}$, which simply means that each neighbour contributes to the similarity in inverse proportion to its number of neighbors. I analyzed the results and compared it with the previous case.

3rd Approach
I realized that since kmeans algorithm can also provide probability of a node belonging to a particular cluster, therefore it provides an opportunity to be able to find overlapping communities.

2 Dataset

I used the dataset(Twitter mentions and retweets) given here(`http://wiki.gephi.org/index.php/Datasets`). I converted the dataset to ".gml" file using Gephi to be used with igraph library(python module). I have uploade the code and the dataset on my public repository(`https://github.com/gauravsc/kmeans_community`).

3 Source Code

All the source was written in python using networkx library and igraph library to compare performance of algorithm against other existing algorithms. The source codes are hosted on the following url (`https://github.com/gauravsc/kmeans_community`). The prerequisites for running the code are given below:

- Pycluster module for python
- Networkx module for python
- igraph library for python
- matplotlib to observe the graphical representation of graphs
- Two dataset files named "twittercrawl.dot" and "twitter.gml"

Dataset files are also present in the repository and can also be downloaded from the following link(`http://wiki.gephi.org/index.php/Datasets`) (Although, You can only download the ".dot" file from the link but not the "gml" file, but you can use the file provided in this repository or convert it yourself using Gephi)

4 Evaluation

The algorithms results were compared against the fastgreedy algorithm using average clustering coefficient value. It basically means, I averaged the value of clustering coefficient for all the clusters obtained using fastgreedy algorithm and kmeans based approach. I provide below the values of average clustering coefficient for different values of number of clusters in kmeans for (average clustering coefficient is denoted by **"avc"** below):

1st Approach: Using simple vector representation

- For k=3, with bag of words approach
  - $kmeans_{avc} = 0.584405$
  - $fastgreedy_{avc} = 0.22581$
- For k=4, with bag of words approach
  - $kmeans_{avc} = 0.22097$
  - $fastgreedy_{avc} = 0.22581$
- For k=5, with bag of words approach
  - $kmeans_{avc} = 0.725481$
  - $fastgreedy_{avc} = 0.22581$

2nd Approach: Modified neighbor information

- For k=3, with modified neighbor information appraoch
  - $kmeans_{avc} = 0.358471$
  - $fastgreedy_{avc} = 0.22581$
- For k=4, with modified neighbor information appraoch
  - $kmeans_{avc} = 0.17063$

  - $fastgreedy_{avc} = 0.22581$
- For k=5, with modified neighbor information appraoch
  - $kmeans_{avc} = 0.22681$
  - $fastgreedy_{avc} = 0.22581$

I present below the graphs for communities detected using kmeans based appraoch with number of clusters=3. As appears from the graph(rough idea) that the communities are very tighly coupled, since the number of links among members of the community are very high, which is also leading to overpopulation of links.

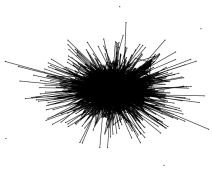

Fig. 1. 1st community

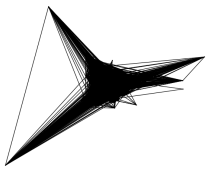

Fig. 2. 2nd community

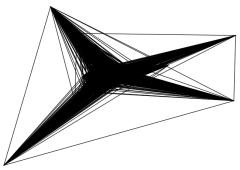

Fig. 3. 3rd community

Both the methods have complexity of approximately $O(nlogn)$ in practical situations.

5 Comparative Analysis

In comparison to the fast greedy algorithm there are number of differences in the proposed appraoch. I will describe some of them below:-

1. The fast greedy algorithm is a heirarchial approach and minimizes modularity function at each step whereas kmeans based appraoch doesn't minimize modularity as an objective function.
2. K-means based approach can be scaled more efficienctly to larger networks since kmeans itself is easily convertible to map-reduce formats.
3. The values of clustering coefficient were generally better or equal in case of kmeans based approach compared to the fast greedy approach.
4. K means based approach requires prior information about number of clusters, which is a disadvantage.
5. Kmeans algorithm is non deterministic and therfore the results may vary with respect to the initial seeds.
6. The kmeans based approach can be used to find overlapping communities since, points that are close to two clusters can be considered to be a part of both clusters. I didn't do any evaluation of this approach, but it can also be an advantage of kmeans based approach. There is no such concept of overlapping communities in fastgreedy community detection algorithm.

6 Reproduction of Results

In order to reproduce the results you just have to run the file "network.py" using the command "python network.py". The datasets("twitter.gml" and " twittercrawl.dot") should be in the same directory as the file "network.py". All the code and dataset files are hosted on the repository(`https://github.com/gauravsc/kmeans_community`)

7 Conclusions

The kmeans based community detection algorithm appears to be promising, as it produces communities with high clustering coefficient. It is also easy to implement and can be extended to large graphs because of its ability to be run in map-reduce mode. It can also work with overlapping communities, although I haven't evaluated that approach. I also wish to tell that while implementing this algorithm and after conceptualising the idea I came across the following paper(`http://link.springer.com/chapter/10.1007%2F978-3-642-16493-4_17#page-1`), although I wasn't able to read the paper as it was behind paid walls. I assume that this paper is much more different and complex than my simple algorithm.