VE444 Project Final Report Group 8

Mohammadali Asgari Vaziri **Shen Yang** **Wang Zhe**

Abstract

An individual's social network is big and consists of different kinds of social circles, such as classmates, close friends, co-workers, etc.. While in real life people usually categorize friends in the social software friend list manually, we want to figure out some way to do this job by computer. This is actually a node-clustering problem. In this project, we reimplement an unsupervised algorithm to detect the social circles by studying on Facebook ego-network, an individual's network of connections with its friends. We detect the circles by studying the features of the connecting nodes, along with the ego-user itself. Here we use SNAP in python language to process the data and realize the algorithm. Finally we compare our result to the ground-truth to see how the method works.

Introduction

In today's society, people are constantly looking for socializing and finding new social networks. These social networks are very wide in scope and variety. Online social networks such as Facebook, Twitter, YouTube and Friendster are the platforms that allow users to follow a series of content that have been generated by thousands of their friends and acquaintances. The amount of content being generated each day is so overwhelming that if is not organized, it would be a huge mess of information overload. Based on the platform's feature, each user tries to organize their network and the content related to them by creating categorized groups that would be referred as social circles in this paper. Once these circles have been clearly defined, there would exist a powerful tool for privacy measures, content filtering, searching and sharing information through the network and better accessibility to different parts of the network.

At the moment, the platforms that are being explored in this paper would have the user to either manually categorize his circles or they would identify friends sharing a common attribute on a broad scale. None of these approaches would be efficient, one is very time consuming and hard to maintain and the other would likely not fit with the user's sense of community and may have very weak execution in case of missing or hidden information.

Here we would try to work on the methods that would be able to automatically discover users' social circles. These circles are user-specific since each user would be organizing their networks independently and in a personalized manner. This gives us the ability to define the problem of circle detection as a clustering problem on user's ego-network. The ego-network is the network of friendships between user's friends. In order to have a clear and comprehensive understanding of the problem we may consider a user as 'u' which we call "ego" and then we identify all the user's friends and will have them as vi's which we refer to them as "alters". Now the goal is to identify and demonstrate the circles to which each alter vi belongs. This means finding nested as well as

overlapping communities/clusters in u's ego-network. A good graphical example of that is shown in Figure 1.

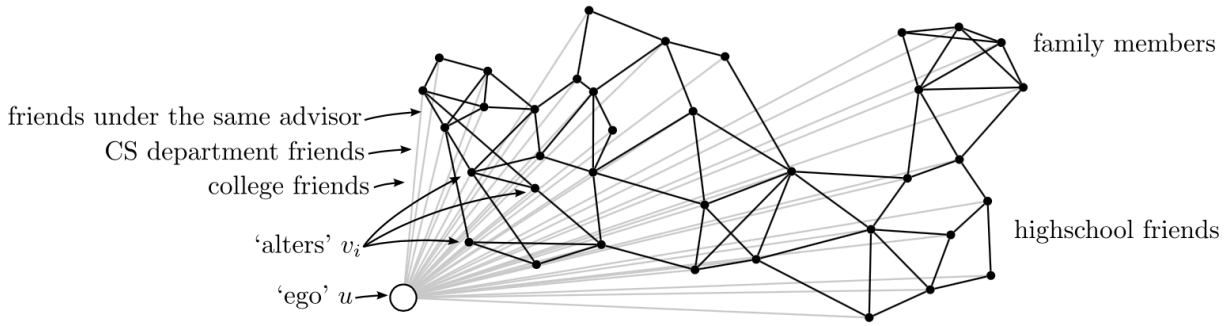

Figure 1: An ego-network with labeled circles.

There are two sets of data that would help us to properly identify the circles, one would be the set of edges of the ego-network which means circles would be formed by "densely-connected sets of alters". The other set of data that attributes to creation of a circle is the "common properties or traits" that alters share in that circle. Circle affiliations would be modeled as latent variable and we would have function of common profile information for similarity between alters.

Related Works

- Julian McAuley and Jure Leskovec, *Learning to Discover Social Circles in Ego Networks*

  This essay is the fundamental source of our project. It describes how to develop a method to predict the social circles of an individual, and provides a theoretical background for our unsupervised method (which will be explained later). It applied the unsupervised method to three social networks: Facebook, Google+, and Twitter, and compared the result with the ground-truth.

- J. Yang and J. Leskovec, *Defining and Evaluating Network Communities based on Ground-truth*

  This essay explores the structural definitions of network communities and examines the performance in identifying the ground-truth. It gives a series of scoring functions to evaluate the likelihood for a given set of connectivity structure of nodes to form community, including scoring functions based on internal connectivity, scoring functions based on external connectivity, scoring functions that combines both, and scoring functions based on a network model.

- F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, *Defining and identifying communities in networks*

This essay discusses the lack of a general and quantitative definition of community. It deals with the problem by showing how quantitative definitions of community are implemented in practice in the existing algorithms. It also proposes a local algorithm to detect communities which is better than most of the existing algorithms.

Datasets Statistics

The datasets we use all comes from Stanford Network Analysis Project[2]. More specifically we focus on three datasets, edo-Facebook, ego-Gplus, ego-Twitter from [1]. Here are some statistics for those datasets.

ego-Facebook

Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

Table 1: Statistics for ego-Facebook

ego-Gplus

Nodes	107614
Edges	13673453
Nodes in largest WCC	107614 (1.000)
Edges in largest WCC	13673453 (1.000)
Nodes in largest SCC	69501 (0.646)
Edges in largest SCC	9168660 (0.671)
Average clustering coefficient	0.4901
Number of triangles	1073677742
Fraction of closed triangles	0.6552
Diameter (longest shortest path)	6
90-percentile effective diameter	3

Table 2: Statistics for ego-Gplus

ego-Twitter

Nodes	81306
Edges	1768149
Nodes in largest WCC	81306 (1.000)
Edges in largest WCC	1768149 (1.000)
Nodes in largest SCC	68413 (0.841)
Edges in largest SCC	1685163 (0.953)
Average clustering coefficient	0.5653
Number of triangles	13082506
Fraction of closed triangles	0.06415
Diameter (longest shortest path)	7
90-percentile effective diameter	4.5

Table 3: Statistics for ego-Twitter

Dataset Related Files

The set includes the five descriptive files for each ego user, whose features are explained below.

- ego.circles

  In this file, the social circles of the ego user is listed line by line. Each line represents a circle, and the nodes in this line are the members of this circle. This file will be used to verify the prediction result.

- ego.edges

  In this file, nodes appear in pairs, indicating the edges between them. The nodes mentioned in this file are all connected to the ego user.

- ego.egofeat

  This file describes the features of the ego user itself, will be used to construct the edge features (since the members of circles should have common relationships to the ego of the ego-network), so that we can better capture the subjective preferences of the ego user.

- ego.feat

  This file describes the features of every node connected to the ego user, which is represented as Fig.2

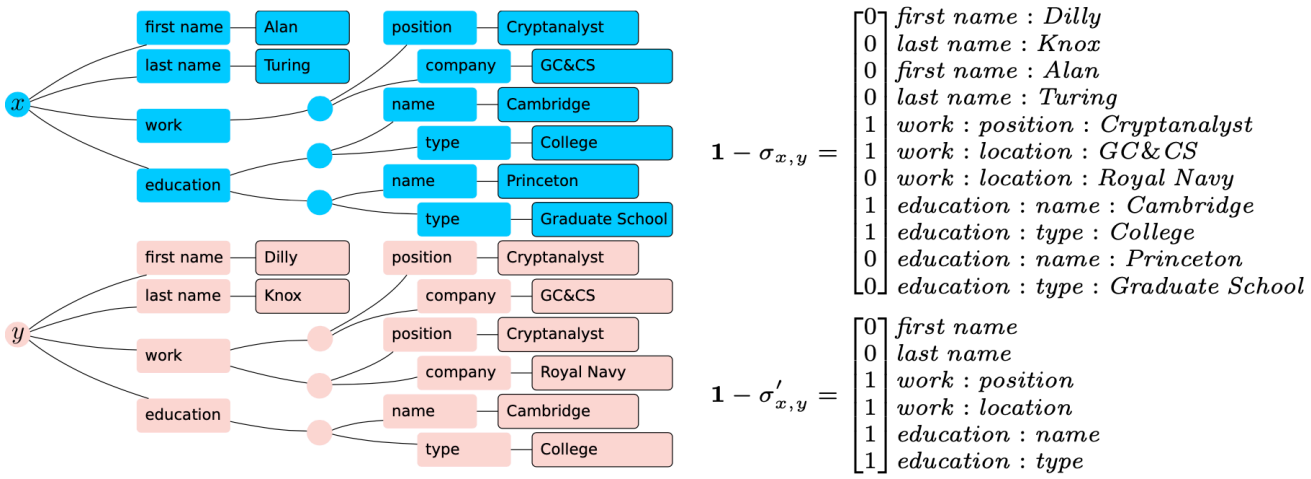

Figure 2: Feature construction, where profiles are stored in tree structures. Analysis can be made by comparing paths of features of two nodes in the network.

- ego.featnames

  This file listed the names of the features which are not shown in the ego.egofeat and ego.feat for convenience.

Theorems and Algorithms

Feature Construction

We are going to apply an unsupervised algorithm to process the data mentioned above. To process the node information in the dataset, we draw a tree for each node, on whose leaves are the characteristics of the node. Fig.2 shows the tree of node x and y.

To construct the feature vector of x and y, use $\sigma_{x,y}$ to denote the binary difference vector of x and y.

$$\sigma_{x,y}[l] = \delta((l \in \mathcal{T}_x) \neq (l \in \mathcal{T}_y))$$

However this vector has a disadvantage of high-dimensional, so it is hard to analyze. Therefore we introduce another difference vector $\sigma'_{x,y}$, which discard the specific values of the common features.

$$\sigma'_{x,y}[p] = \sum_{l \in \text{children}(p)} \sigma_{x,y}[l]$$

Based on the difference vectors, we construct the edge features $\phi(x, y)$. According to the properties of the social circles, firstly if x and y are in the same circle, they should have similarities.

$$\phi^1(x, y) = (1; -\sigma_{x,y})$$

Secondly, x and y must have features in common with the ego user.

$$\phi^2(x, y) = (1; -|\sigma_{x,u} - \sigma_{y,u}|)$$

Similarly for 'compressed' difference vector $\sigma'_{x,y}$, we have

$$\varphi^1(x,y) = \left(1; -\sigma'_{x,y}\right)$$

$$\varphi^2(x,y) = \left(1; -|\sigma'_{x,u} - \sigma'_{y,u}|\right)$$

Probability of Edge Existence

In an attempt to create a model for friendships in social circles, the inputs of an ego-network is defined as a mathematical function of $G = (V, E)$ and then models the probability that a pair of nodes $(x, y) \in V \times V$ form an edge which would result to

$$p((x,y) \in E) \propto \exp\{\underbrace{\sum_{C_k \supseteq \{x,y\}} \langle \phi(x,y), \theta_k \rangle}_{\text{circles containing both nodes}} - \underbrace{\sum_{C_k \not\supseteq \{x,y\}} \alpha_k \langle \phi(x,y), \theta_k \rangle}_{\text{all other circles}}\}$$

Where $\langle \phi(x,y), \theta_k \rangle$ for an edge is high if both nodes belong to the circle C_k, and low if either of them do not. The model parameters set for all the circles from 1 to k is defined as

$$\Theta = \{(\theta_k, \alpha_k)\}^{k=1\ldots K}$$

We would like to modify Θ as well as C_k in the process of the model training, for the given network. Finally, we now introduce our loss function for the model, the log-likelihood function.

Log-likelihood Function

Here is the loss function we used to evaluate the model, we use the log likelihood function to determine the likelihood for the cluster(social circle) to form. In practice, we want to maximize the log-likelihood function to get the circles with highest probability to form.

$$l_\Theta(G; \mathcal{C}) = \sum_{e \in E} \Phi(e) - \sum_{e \in V \times V} \log\left(1 + e^{\Phi(e)}\right)$$

where $\Theta = \{(\theta_k, \alpha_k)\}^{k=1\ldots K}$ is the set of parameters, with notation

$$d_k(e) = \delta\left(e \in C_k\right) - \alpha_k \delta\left(e \notin C_k\right), \quad \Phi(e) = \sum_{C_k \in \mathcal{C}} d_k(e) \langle \phi(e), \theta_k \rangle$$

Note the delta function calculates the number of edges in/not in C_k, and $\Phi(e)$ takes the sum of the product with the theta function mentioned before. For the log-likelihood, the first part, $\sum_{e \in E}$, defined the likelihood of the edges that actually appeared in the graph, which is positive. The second part, $\sum_{e \in V \times V} \log\left(1 + e^{\Phi(e)}\right)$, however, defines all edges in the extended complete graph, including those did not actually form. That is why the second part is negative, because we want to eliminate the influence of the potential edges that may form but not.

Model and Algorithm

Overall Process

Here we have used a randomized approach to optimize the performance. For the model parameters $\Theta = \{(\theta_k, \alpha_k)\}^{k=1...K}$, we set the parameter with random initial values, then train the model with the randomized set of parameters. After the training is complete, we record the result and the corresponding loss function(the log-likelihood function), then we reset the parameters and repeat this process. The number of times we repeat this process is defined by a variable in the main function, it is 3 by default, while the value could be further increased. In general, the more times we repeat, the higher chance we could get the result with a higher likelihood.

Model Training

Since the goal is to maximize the log-likelihood, we apply **gradient ascent** method, and use **QPBO**(Quadratic Pseudo Boolean Optimization) to solve the min-cut problem and reduce the circle nodes each time. The following figure shows an example of QPBO solving a simple min-cut problem.

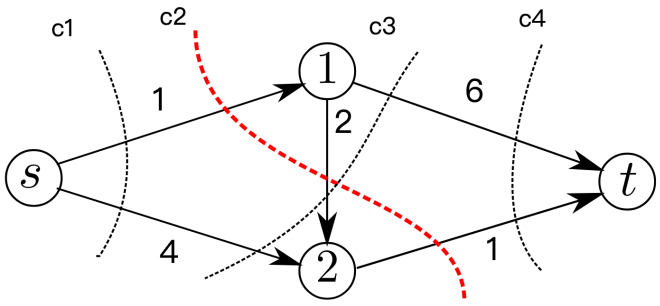

Figure 3: simple example of QPBO on minimum cut

x	$\Phi(x)$
0 0	5
0 1	2
1 0	12
1 1	7

Pseudo Code

The psuedo code for our algorithm can be described as follows:

Algorithm 1: Detecting Social Circle

Input : an ego-network, directed or undirected, with features and names

Output: high-likelihood social cycles within the network

```
1  Function SocialCircle(edge, ego-features, features, names):
2      C ← Cluster.Initialize();
3      C ← ReadFromFile(edge, ego-features, features, names);
4      bestll ← 0;
5      for i ← 1 to random restarts do
6          C.ClusterTrain();
7          ll ← C.LogLikelihood(C.theta, C.alpha, C.chat);
8          if ll > bestll then
9              bestll ← ll;
10             bestClusters ← C.chat;
11             bestTheta ← C.theta;
12             bestAlpha ← C.alpha;
13         end if
14     end for
15     return bestll, bestClusters, bestTheta, bestAlpha;
16 end
17 Function ClusterTrain(C):
18     for i ← 1 to repeats do
19         if First Iteration then
20             C.weights, C.theta, C.alpha, C.chat ← RandomInitialize();
21         end if
22         order[] ← RandInt();
23         for j ← 1 to C.k do
24             o ← order[j];
25             C.chat[o] ← MinimizeGraphcuts(o);
26         end for
27         C.ll ← LogLikelihood(C.theta, C.alpha, C.chat);
28         C.GradientAscent(C.ll, C.theta, C.alpha)
29     end for
30 end
```

Here MinimizeGraphcuts() function is implemented with QPBO.

Result

After implementing aforementioned algorithm, we get clusters and corresponding theta values, which are crucial in detecting social circle. We test our algorithm on 5 egos from Facebook, however, for the sake of convenience, we use ego 698 from Facebook for the purpose of illustration.

In each round of implementation, our algorithm will return a cluster of nodes involved in at least one social circle. Depending on the choices of input K, our algorithm will return K possible clusters. For instance, for K = 12, we get 12 clusters in total, and one particular cluster we get is, [1, 5, 6, 10, 11, 23, 24, 27, 28, 30, 33, 37, 38, 40, 41, 42, 44, 45, 46, 47, 50, 52, 53, 56, 57, 58, 59, 61, 62], indicating social circles are formed from these users. Users not listed are considered not belonging to any social circles. Then, we visualize the nodes in each proposed social circle and demonstrate the result with nodes obtained from two rounds of implementation as shown in the following figure. For each group of figures, the figure on the left is the social circles in practice, while the figure on the right is the nodes predicted to belong to at least one social circle by our model. It could be told that our model reaches the requirement of finding most of the nodes belonging to any social circle correctly. Nevertheless, some isolated nodes in the peripheral are also included in our model. Therefore, more tests and validations are required.

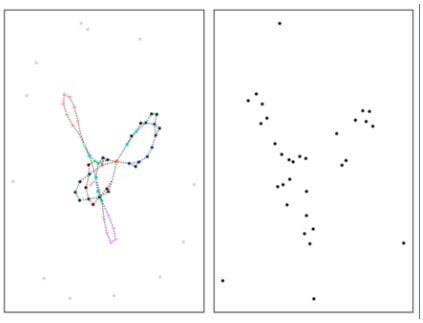

Figure 4: fig1

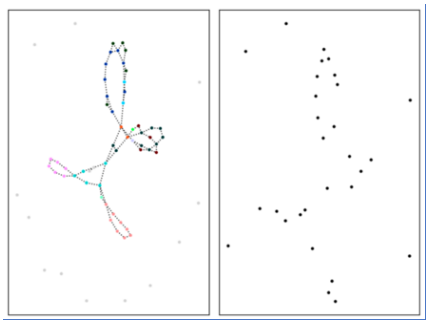

Figure 5: fig2

Figure 6: Comparison of circles in reality and nodes involved in at least one social circle predicted by our model

We first aggregate the results we get to reduce the chances of making mistakes in one particular round of implementation. We add up the number of times each node appears in all K iterations. In our particular example, K = 12. The result we get is shown in the next figure. Different color represents different number of times a specific node appears, and nodes with light color appear less frequently than nodes with dark colors. It is still not clear, however, which proportions of nodes are formed due to which features in the source data.

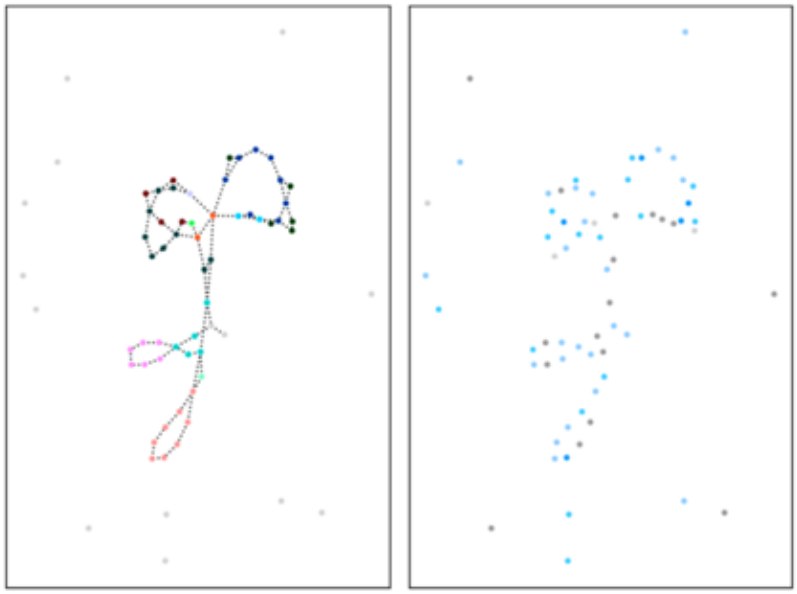

Figure 7: An aggregation of the number of times each node appears in our model

In other words, we want to decide which parts of the nodes should be connected within one particular circle due to which particular feature. We implement this with the output values of theta. In each iteration, we get a list of theta with (number of features +1) values. As theta usually range from (0, 1) in our model, we first choose to set the boundary condition at the level of 0.5. In other words, feature with corresponding theta greater than 0.5 are considered to be significant. However, in real implementation, we find out that 0.5 is not the optimal choice, as it includes redundant information. After trial and errors, we set level to be 0.6 to decide which features are significant in forming social circles. We implement it by first filtering out all features with theta smaller than 0.6. Then, we identify the nodes possessing corresponding features. In our particular case, corresponding significant features are listed in Figure 3. From Figure 3, it can be told that even though the output for each iteration might be different, the significant features are almost the same. In our case, feature 13, 15, 27, 28, 34, 35 appear in all 12 iterations. Therefore, we can conclude aforementioned 6 features are very important for the formation of social circles.

```
[8, 13, 15, 27, 28, 34, 35]
[13, 15, 27, 28, 34, 35, 38]
[3, 13, 15, 27, 28, 34, 35]
[4, 13, 15, 27, 28, 34, 35]
[13, 15, 16, 27, 28, 34, 35]
[13, 15, 22, 27, 28, 34, 35]
[9, 13, 15, 27, 28, 34, 35]
[13, 15, 27, 28, 34, 35, 42]
[13, 15, 18, 27, 28, 34, 35]
[11, 13, 15, 27, 28, 34, 35]
[5, 13, 15, 27, 28, 34, 35]
[13, 15, 27, 28, 34, 35, 45]
```

Figure 8: Features that are significant when theta is chosen to be greater than 0.6

References

[1] Julian McAuley and Jure Leskovec. “Learning to Discover Social Circles in Ego Networks”. In: ().

[2] *Stanford Network Analysis Project*. URL: http://snap.stanford.edu/.