Invisiable Edge
CS 526 Challenge Problem 3, Group 11

Martin, John Urffer, Matthew Wan, Lipeng

December 5, 2012

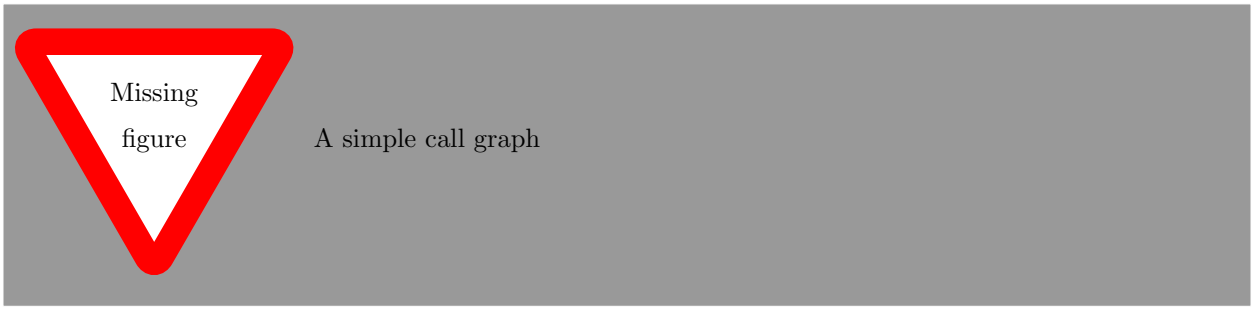
Todo list

1 Introduction

2 Methods

The call graph was provided as a labeled edge list, and was modeled as a weighted, undirected, multigraph The graph was organzied as each of the nodes being the unique identifier to a cutomer, with edges between the customers representing:

- `days` - the number of distinict days that the two nodes communicated during the month,
- `calls` - the number of distinct calls that the the nodes made during the month,
- `secs` - the cummulative sum of all calls that the modes made during the month and,
- `texts` - the number of distinct texts that the two nodes made during the month.



A simple call graph

2.1 LSA Based Similarity Measure

Based on some work we did a few years ago looking at personality profile matching using an LSA based system, I wondered if the premise might hold that a node (person) could be effectively "described" by the set of other people they were connected to. This description could then be measured for similarity to other node descriptions and hopefully we would find that "similar" nodes were connected. In the past work we did there was some promise in the initial results for group analysis, but it was not pursued very far and there were other data items that were being considered as well (and it had nothing to do with telephone usage patterns).

To test the theory, I initially constructed a sparse matrix for each of the graph files for both the Moria and Standelf. For each connected node, the connection weight was determined by taking each of the 4 attribute values, dividing by thier respective standard deviation, and summing the results into a single weight for each connection. I also reran the experiment using a simpler weighting scheme giving a 1 if any call was made and another 1 if any text was made.

This sparse matrix in any case was extremely sparse, being around 0.00051% nonzero for Moria and 0.00059% for Standelf. We usually deal with matrices that have a nonzero rate of 0.001% to 0.01% for text mining applications. I computed a truncated SVD for these sparse matrices, forming an LSA space at approximately 250 dimensions for each. Then using these 250 dimenstion spaces I looked at comparing the first 1,000 nodes of each graph to all the other nodes in the graph computing their vector cosine similiarity and noting if the nodes were connected or not. This takes a while to run so there has not been much opportunity to tweak any of the parameters, therefore I cannot conclude with any certainty, but the initial results do not look promising. The first sets that I processed did not show any clear indication of connectivity determined by the similarity measure.

2.2 Artifical Neural Network Classification

The next approach implemented was an artifical neural network classification system. Using the `PyBrain` tool module a feed forward neural network was constructed with 8 inputs represting the edge between node `u` and `v`:

- the degree of node `u`,
- the closeness of node `u`,

- `days` of the edge,
- `calls` of the edge,
- `secs` of the edge,
- `texts` of the edge,
- the closeness of node `v`, and,
- the degree of node `u`.

The closeness of node was calucated using the `networkx` module with `closeness_centrality`, defined to be 1 over the average distance to all other nodes. The distances where not weighted, and all distances where normalized by the graph size. The degree was calculated with `degree` as the the sum of the edge weights of adjacent nodes for a particular node (completed for all nodes). The other elements of the training vector were simply filled with the values from the edge.

Training data for the neural network was then the set of all example nodes and edges presented in the target class of 1 (the edge exists). This was ultimately a flaw in the design of the neural network (or any classification based system) as discused in Section 3.

2.3 Spectral Approach for Finding Possible Communities in Graph

For this call graph data set, an interesting question comes to our mind which is how to find the community structure in the graph. The cell phone users in the same community will have stronger relationship between each other than that between uses from different communities. Therefore, if we can find such communities in the call graph, it might be very helpful for us to understand the cell phone users' behavior.

The approach we use to find community structure in call graph is named spectral graph. The basic idea of this approach is to use the eigenvectors of the Laplacian matrix of the graph to partition the vertices into different clusters. Now we are going to introduce the definition of the Laplacian matrix of a graph.

In graph G, let d_v denote the degree of vertex v, the Laplacian of G is defined as the matrix

$$L(u,v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0, \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Let T denote the diagonal matrix with the (v,v)-th entry having value d_v. The Laplacian of G can be written as

$$L = I - T^{-1/2} A T^{1/2} \tag{2}$$

where A is the adjacency matrix of G (i.e., $A(x,y) = 1$ if x is adjacency to y, and 0 otherwise) and I is an identity matrix. All matrices here are $n \times n$ where n is the number of vertices in G.

Now let's show how to use spectral graph approach to find possible communities in graph. For example, we first import the raw data of call graph of Moria city in month 1 and generate the adjacency matrix of this call graph as shown in Fig. 1.

From Fig. 1, we can observe that this call graph is very dense and it is impossible for us to find any community structures in this figure because the vertices of this call graph are randomly arranged in the adjacency matrix.

Now let's investigate what the second smallest eigenvector of the Laplacian matrix tells us about this graph. The following MATLAB code is used to compute the eigenvalues and eigenvectors of the Laplacian matrix of call graph of Moria city in month 1.

```
Moria_1_I = speye(size(Moria_1_adj,1));
Moria_1_T = diag(sum(Moria_1_adj,2).^(-0.5));
Moria_1_L = Moria_1_I-Moria_1_T*Moria_1_adj*Moria_1_T;
[Moria_1_V,Moria_1_D] = eigs(Moria_1_L);
```

Then we sort the second smallest eigenvector and permute the vertices of this call graph based on the order of components in the sorted second smallers eigenvector.

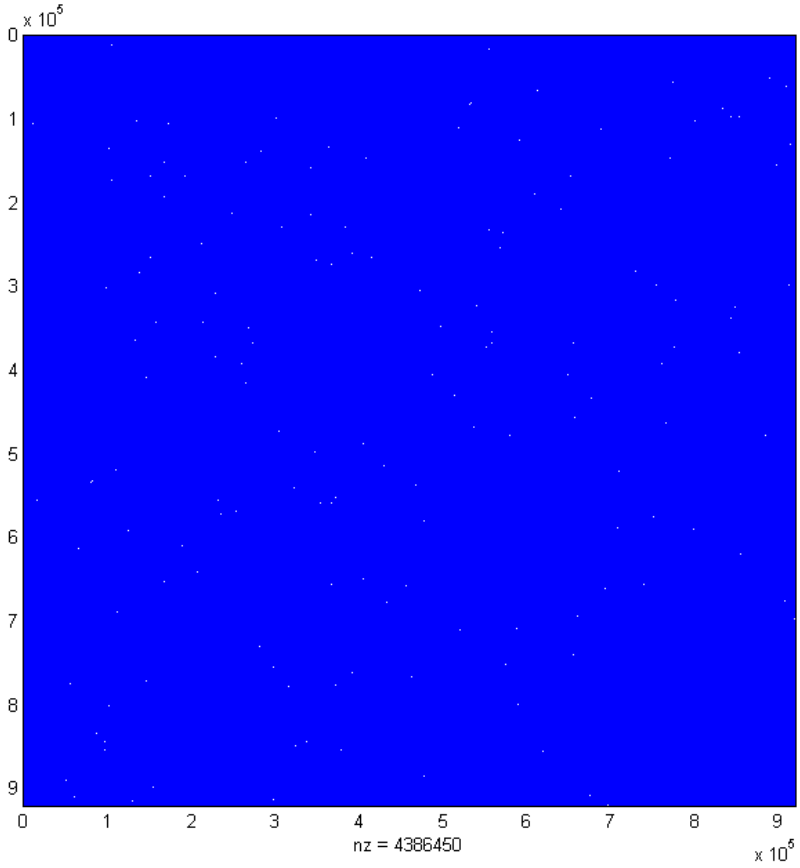


Figure 1: Adjacency Matrix of Call Graph of Moria City in Month 1

```
[ignore Moria_1_p] = sort(Moria_1_V(:,2));
spy(Moria_1_adj(Moria_1_p,Moria_1_p));
```

In the end, the **spy()** function will plot the rearranged adjacency matrix of the call graph, which is shown in Fig. 2.

In Fig. 2, we can observe some community structure, for example, the vertices at the upper left corner of the matrix have stronger connection between each other. Similarly, we can also generate the rearranged adjacency matrix of call graph of Moria city in month 2, as shown in Figure 3

From Figure 3 we can observe that though some edges have been removed in the call graph of Moria city in month 2, the community structures in the call graph of Moria city haven't been changed much. In Figure 3, we still can find the community structures which are similar to those in Figure 2. For instance, the vertices at the lower right corner in Figure 3 and the vertices at the upper left corner in Figure 2 are in the same community.

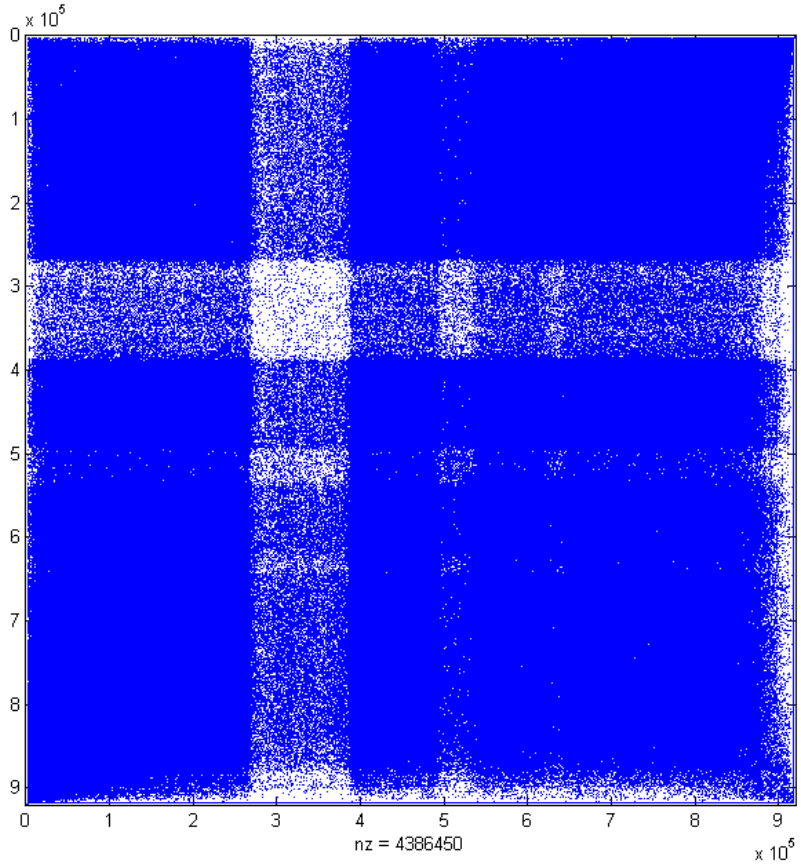


Figure 2: Rearranged Adjacency Matrix of Call Graph of Moria City in Month 1

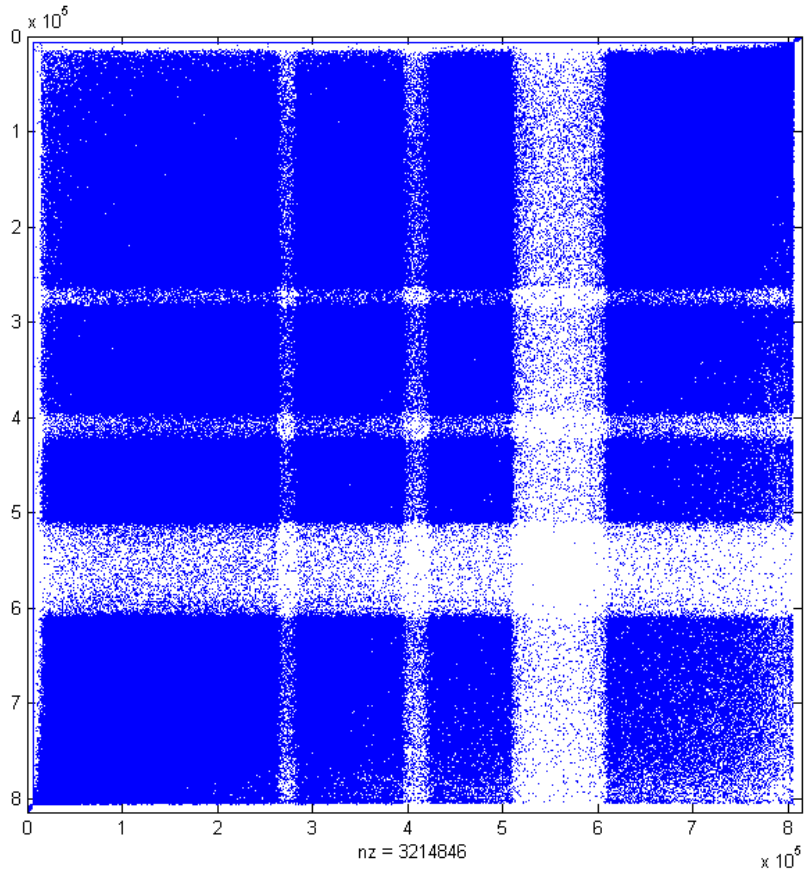


Figure 3: Rearranged Adjacency Matrix of Call Graph of Moria City in Month 2

3 Resuls

3.1 LSA Based Similarity Measure

3.2 Artifical Neural Network Classification

The artifical neural network trained to a very low (less than 1%) error using 5-fold cross valdiation and back-propogation in 20 itterations. However, when tested the accuracy of predicting wether a node existed or not was 51%. This low accuracy is no better than flipping a coin. This is because the neural network was not feed any negative class examples; all the network had to learn was that an example input pattern corresponded to an edge. Negative input patters could not be created as network input because there is no guarentree that the input pattern would not exists, thus the network could be taught incorrect data. I

4 Conclusions

It is concluded that due to the lack of negative training examples (edges that should not be there) any classification based system will preform with an accuracy of 50%.

5 Acknowledgements

The work was distributed as follows. John completed the analysis of the data with the LSI based similarity measure. Matthew completed the distributions of the data as well as the neural network classification system. He also implemented the probability based approach.