Determining Key Information Carying Nodes in a Social Network using Eigenvector Centrality

Kris Harbold

December 10, 2013

Abstract

Social Networks provide a mechanism for quickly and efficiently dispersing information to large groups of people. It has been shown with the advent of Twitter, Facebook, and other such network, news of global events can make it through network faster than news sources can even publish on events. For this reason Social Networks can be leveraged to spread news for the benefit of companies such as advertisers. By finding the central nodes in a network, information can be delivered to these nodes first; consequently, news can reach more distant nodes, and travel through the most nodes, in the shortest amount of time.

1 Introduction

Social Networks often contain the interests of their users which can be used to find what's similar about nodes that may not be friends, thereby making it possible to isolate leaders and central points within the network. By consolodating a network into an adjacency matrix it can be used to find the eigenvector centrality for each network node. These values can then be used to choose those nodes that are best for dispersing information to the entire network.

In this project, a network could be either read from a file or built by the user. Between zero and five interests are assigned to each network node. Matrix multiplication is preformed to create an adjacency matrix representing the similarity between users. Eigenvector centralities are calculated on the resultant matrix. The eigenvector and user ID vector are sorted with the centrality values in decending order. After the user specifies n nodes to disperse information to, the program prints the top n nodes and their UID.

2 Social Network Simulation

2.1 Reading From File

An option is offered to the user to read the user IDs from a file. UIDs are read from the first column of the file and stored into an integer array. The values of the array are the UIDs from the file.

2.2 Building Network

The user can also simulate their own social network by adding nodes to the network by either specifying friends or creating new nodes. The user can switch the reference node for building the network, making it possible to add friends to any node. Nodes within the social network are stored in a linked list. Each node in the network has a binary search tree for storing friends of the user. When creating a friend, a new node is created in the linked list. The new friend is added to the current node's friend tree and the friend's node has the current user added to their friend tree.

To convert the graph into the necessary UID array, each user name is hashed using a hash function into an 8 digit user ID. An array is created, each array value is the UID of one node of the network graph.

3 Adjacency Matrix

Zero to five interests are assigned to each node randomly to save time, this could be choosen by the user. A matrix where $I_{i,j}$ is 1 when the user i is interested in j interest. This matrix is multiplied to its transpose as described in Eq. 1 resulting in a User by User matrix.

$$U = I \times I^T \tag{1}$$

4 Eigenvector Centrality

The User by User U matrix is used to calculate the centrality of each node within the network. The closer the resulting value is to one the more central that node is in the network.

Eigenvector centralities are calculated by finding the degree of centrality vector. This vector is created using Eq. 2 where d is the degree of the vector and x is a $|1 \times m|$ vector filled with ones, m being the size of the UID vector.

$$D(d) = U^d \times x \tag{2}$$

The eigenvectors are calculated by taking this vector and dividing each value by the magnitude of the degee vector as shown in Eq. 3.

$$Eigenvector = \frac{D(d)}{|D(d)|} \tag{3}$$

In which the magnitude is calculated as follows:

$$|D(d)| = \sqrt{d_0^2 + d_1^2 + ... + d_m} \tag{4}$$

Choosing the d as some large value gives an approximate of the centrality.

5 Place Ads

Merge sort is used to order the eigenvectors in decreasing order. UIDs are sorted alongside to ensure UIDs still line up with their centrality. The top n user specified UIDs and eigenvector centrality values are printed for the user.