CS 225 Project Results

Team Members: Bowen Xiao, Richard Przybek, Nathen Smith, Vijay Gopalakrishnan

Overview

In our project we decided to use the Twitter Ego dataset and implement Djikstra's and Betweenness Centrality algorithms to analyze it. In the dataset we found there were sets of five files and each set of files corresponded to one twitter user and the other users they follow. For each set we could construct a graph corresponding to a network of users that the main user follows. Each vertex in our graph represents a user, and an outgoing edge from A to B tells us A follows B on Twitter. Every user comes with a set of features corresponding to @'s or #'s they have used in their tweets. We assign edge weights based on the number of common features between two users.

Dijkstra's algorithm tells us the shortest path between two users based on the number of tweets they have in common with their followers. This can be thought of as a chain of retweets or a path of influence from the users someone is following. The Betweenness Centrality algorithm tells us how many chains go through a user. This can be thought of as how influential someone is on other users.

Run-time Analysis

Algorithm	Time Complexity
FillGraph()	$\|E\| + \|E\|\|F\| + \|V\|\|F\|$
DFS	$\|V\| + \|E\|$
Dijkstra's	$\|E\|+\|V\|\log(\|V\|)$
Betweenness Centrality	$\|V\|^2\log(\|V\|)$

V = vertices, E = edges, F = features

Problems Encountered and Design Choices

The first design issue we encountered was how to assign weights to the edges of our graph based on the Twitter dataset. We discussed using relationships between other vertices and features but opted for common features between users since it would tell us how much impact a user has and it is more unique to the twitter dataset (other social media has basic following/not following).

Another issue we had was how to account for the dataset being directed and figuring out what our methods would return in the Graph class. We chose to return a path from Djikstra's algorithm and DFS. Betweenness centrality returns a measure of a user's centrality.

While we were researching the betweenness centrality algorithm, we realized that we needed to find all of the shortest paths between all of the vertices. Initially, we thought we could use Dijkstra to do this, but our implementation of Dijkstra only returned one of the possible shortest paths, in others words it ignored tiebreakers. We attempted to modify Dijkstra to output all of the shortest paths between two vertices, but that task proved to be unexpectedly difficult. In the end, we compromised and implemented betweenness centrality using our current Dijkstra algorithm assuming there is one shortest path for every pair of users. This means that betweenness centrality will output a minimum centrality value less than or equal to the real centrality measure.

Testing

To ensure the correctness of our algorithms, we built different tests of various complexities. For Dijkstra's algorithm, we implemented test cases where there were no outgoing edges, see "Dijkstra - Path doesn't exist.png" for the specific graph we tested it on. Since no paths exist between the source and destination vertices, we needed to return an empty vector containing no path. Another case we needed to consider when testing Dijsktra's was when the graph is disconnected, see "Dijkstra - Disconnected graph.png". Here, we needed to make sure that the correct nodes are visited corresponding to the edges and Dijksta's shortest path should work the same as a connected graph. We needed to perform similar, specific-case tests on our DFS as well. These include the ability to traverse a cyclic graph, a disconnected graph, and a no-outgoing path graph. For the betweenness centrality algorithm, we included similar graphs, involving cycles for additional complexity. The betweenness centrality function takes in a vertex and returns an int representing the number of paths going through that vertex, so we needed to make sure the functionality does not vary between different types of graphs. More detailed information regarding the test cases can be found in the 'tests' folder of our repository.