Project 1 – Fast Computation of User Centrality Measures

Student Name: Eirik Dahlen
Student ID: 46301282
Student Email: e.dahlen@uq.net.au

In this report I've created a program that successfully calculate the node betweenness centrality score and the PageRank score for an undirected graph based on anonymized Facebook social network data. The data contains 4039 nodes, 88234 edges in total. Each line of the data represents an undirected link starting from one node to another.

Task 1 - Betweenness Centrality

Betweenness centrality measures the number of times a node lies on the shortest path between other nodes. This measure shows which nodes are 'bridges' between nodes in a network. It does this by identifying all the shortest paths and then count how many times each node is involved in a shortest path ([1] Cambridge Intelligence, 2020)

The implementation of betweenness centrality is based on the Brandes algorithm ([2] Utrecht University, 2018).
In short, the steps of the algorithm is the following:
1) Iterate through all nodes in the network
2) Do breadth-first-search for each node, calculate distances and shortest paths count, push the nodes to a stack
3) Visit nodes in stack in a backpropagating manner, and calculate betweenness centrality
4) Normalize results, optional

The formula used for calculating the betweenness centrality is based on the one from lectures in INFS7450:

$$\delta_s(v_i) = \sum_{w:\ v_i \in pred(s,w)} \frac{\sigma_{sv_i}}{\sigma_{sw}}(1 + \delta_s(w))$$

Here, σ_{sv_i} is number of shortest paths between nodes s and w that pass through v_i, while σ_{sw} is the number of shortest paths between nodes s and w in total.
$\delta_s(v_i)$ is the dependence of s to v_i. There are a cubic number of pairwise dependencies like this. The smart thing about Brandes algorithm is that the dependencies can be aggregated without calculating them all explicitly by calculating dependency of s on v_i based on dependencies one step further away ([3] University of Cambridge, 2017)

The results of the calculation are evaluated towards the calculation done in NetworkX ([4] GeeksforGeeks)

Rank	My results		NetworkX results	
	Node	Value	Node	Value
1	107	0.4805180785560147	107	0.4805180785560147
2	1684	0.33779744973020004	1684	0.33779744973020004
3	3437	0.23611535735892794	3437	0.23611535735892794
4	1912	0.22929533958687848	1912	0.22929533958687848
5	1085	0.14901509211665154	1085	0.14901509211665162
6	0	0.14630592147442847	0	0.14630592147442847
7	698	0.11533045020560784	698	0.11533045020560784
8	567	0.09631033121856238	567	0.09631033121856238
9	58	0.08436020590796656	58	0.08436020590796656
10	428	0.0643090623932386	428	0.0643090623932386

Comparing my results to the NetworkX implementation, they give the same ranking of nodes, as well as almost identical calculations. I also had a timer in my program to see how long it took to terminate. The NetworkX calculation was superior in regards to time complexity, as it used around 240 seconds, while my implementation used somewhere between 400 seconds. I think this has to do with what kind of data structures was used, and how built-in Python functions works in terms of lookup and search.

Task 2 - PageRank

PageRank is one of the ranking algorithms behind the original Google search engine, and assign nodes a score based on their connections, and their connections' connections ([1] Cambridge Intelligence, 2020). PageRank prevents that nodes which becomes an authority (node with high centrality), passes all its centrality along all of its out-links. With an analogy we can see why: not everyone known by a well-known person is well-known. Preventing this is achieved by dividing the value of passed centrality by the number of outgoing links, i.e., out-degree of that node.

The formula for calculating the PageRank score is described as:

- Set $\boldsymbol{c}^{(\boldsymbol{0})} \leftarrow 1, k \leftarrow 1$
- **1:** $\boldsymbol{c}^{(\boldsymbol{k})} \leftarrow \alpha A^T D^{-1} c^{k-1} + \beta 1$
- **2:** If $\left\|\boldsymbol{c}^{(\boldsymbol{k})} - \boldsymbol{c}^{(\boldsymbol{k-1})}\right\| > \varepsilon$:
- **3:** $k \leftarrow k + 1$, goto **1**

The highlighted orange area is the matrix calculation. α is a damping factor, set as 0.85 in this project, and β is a personalization factor, set as 0.15 (or 1 - α). I did not consider dangling nodes (a node with an out-degree of 0) as this was an undirected graph and the problem would never be relevant.

The algorithm in my code is based on the implementation from the lecture in INFS7450, where power iterations is used.
In short, the steps of the algorithm is the following:
1) Assign adjacency matrix as a dict of dicts: this store a node as key and its neighbours/outgoing links as values in the dictionary. That results in a sparse matrix.
2) Create degree matrix as a dictionary and invert it, also a sparse matrix
3) Create c matrix where the entries are 1.0
4) Do iterations of pagerank where the next c is calculated based on previous c and the neighbours' outgoing links, as well as alpha and beta
5) Stop when the L2 norm is smaller than the tolerance
6) Normalize the results, optional

The results of the calculation are evaluated towards the calculation done in NetworkX ([5] NetworkX Documentation, 2019):

Rank	My results		NetworkX results	
	Node	Value	Node	Value
1	3437	0.007614586844749603	3437	0.007614586844749602
2	107	0.006936420955866111	107	0.006936420955866113
3	1684	0.006367162138306836	1684	0.006367162138306825
4	0	0.0062896026184665445	0	0.006289602618466542
5	1912	0.003876971600884491	1912	0.0038769716008844957
6	348	0.002348096972780578	348	0.002348096972780577
7	686	0.0022193592598000193	686	0.0022193592598000193
8	3980	0.002170323579009992	3980	0.0021703235790099928
9	414	0.0018002990470702253	414	0.0018002990470702264
10	698	0.001317115313836881	698	0.0013171153138368812

Comparing the results of the NetworkX implementation and my own implementation, we see that the ranking of the nodes are identical. The values of each node is almost identical as well, with minor differences in the rightmost decimals.
The power iteration method and has no guarantee of convergence. The way that the threshold for stopping the power iteration is calculated is different in the two approaches: NetworkX is using the L1 norm, while I'm using the L2 norm. My tolerance is 1.0^{-4} while they

use 1.0^{-6}. Both multiply this tolerance with N (number of nodes in the network). This helps make the threshold condition to be adaptive, and scale according to the size of the input-network. One approach to find where the threshold should be is to set max_iterations as a high number and see when the value of L2 norm converges.

Summary

This project has unveiled that different node measurements can give different top-nodes in a network. Calculating Betweenness Centrality and PageRank gives me two sets of nodes, where the intersection of these sets gives six nodes that appear in both top-10 results. Another thing that is worth noticing is that the ranking of these six nodes is different in the two calculations; for betweenness the node 107 is ranked number 1, while node 3437 is for PageRank. It is clear that to gain insight in your social network, several centrality measures could and should be used. Another valuable lesson from this project is that you can find the best fitting measurement for your task, by combining the knowledge of your algorithms and the domain knowledge about your network.

Reference

[1] Cambridge Intelligence: *Social network analysis 101: centrality measures explained*.
2nd January 2020, Andrew Disney
URL: https://cambridge-intelligence.com/keylines-faqs-social-network-analysis/

[2] Utrecht University: *Using preprocessing to speed up Brandes' betweenness centrality algorithm.* January 2018, Steven Fleuren
URL: https://www.cl.cam.ac.uk/teaching/1617/MLRD/slides/slides13.pdf

[3] University of Cambridge: *Betweenness Centrality*
2017, Ann Copestake and Simone Teufel
URL: https://www.cl.cam.ac.uk/teaching/1617/MLRD/slides/slides13.pdf

[4] GeeksforGeeks: *Betweenness Centrality (Centrality Measure)*
Unknown, Jayant Bisht
URL: https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/

[5] NetworkX Documentation: *pagerank_alg*
Oct 17th 2019, NetworkX Developers
URL:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html