Project № 1

INFS7450: Social Media Analytics

William E. G. Kvaale

46301303

w.kvaale@uq.net.au

Abstract

In this project I have experimented with Network X, Betweenness Centrality and Page Rank. To calculate the Betweeness Centrality of the given Facebook Dataset, I have implemented Brandes Algorithm [1]. For the PageRank implementation I used the original paper [2] as my baseline. However, I also accomodated for the advise given in the tutorials regarding the L2 norm instead of the L1 norm.

To prepare the data the function `nx.read_edgelist(path)` from NetworkX [3] was used to read a text file composed of edges, and return a built graph with vertices (nodes), edges (links) and neighbors.

Task 1: Betweenness Centrality

Calculating the Betweenness Centrality can be formulated through the following equation

$$C_b = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_s t} \tag{1}$$

Here $\sigma_s t$ is all the shortest paths from node s to t, and $\sigma_s t(v)$ is all of those shortest path, that also runs through v.

Given an unweighted and undirected graph, Brandes' Algorithm takes $\mathcal{O}(|V||E|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges, respectively. Since we are operating on an unweighted graph, we run a Breadth-First Search (BFS) instead of Dijkstra to find the shortest paths.

By following the original paper by Ulrik Brandes the following top 10 nodes was found:

Rank	Node	C_b
1	107	0.480518
2	1684	0.337797
3	3437	0.236115
4	1912	0.229295
5	1085	0.149015
6	0	0.146305
7	698	0.115330
8	567	0.096310
9	58	0.084360
10	428	0.064309

Table 1: Results using Brandes Algorithm

But what is *Betweennes Centrality* (C_B) a measure for? Having a node with high C_B removed, can for instance disconnect graphs. This is one way to look at it. In simple terms, it measures the degree to which a node is between other nodes. This we can see in Figure 1 below. Here the top 10 nodes are the only visible nodes, but all the edges are still visible. One can observe that we find all the top 10 nodes *in between* other nodes.

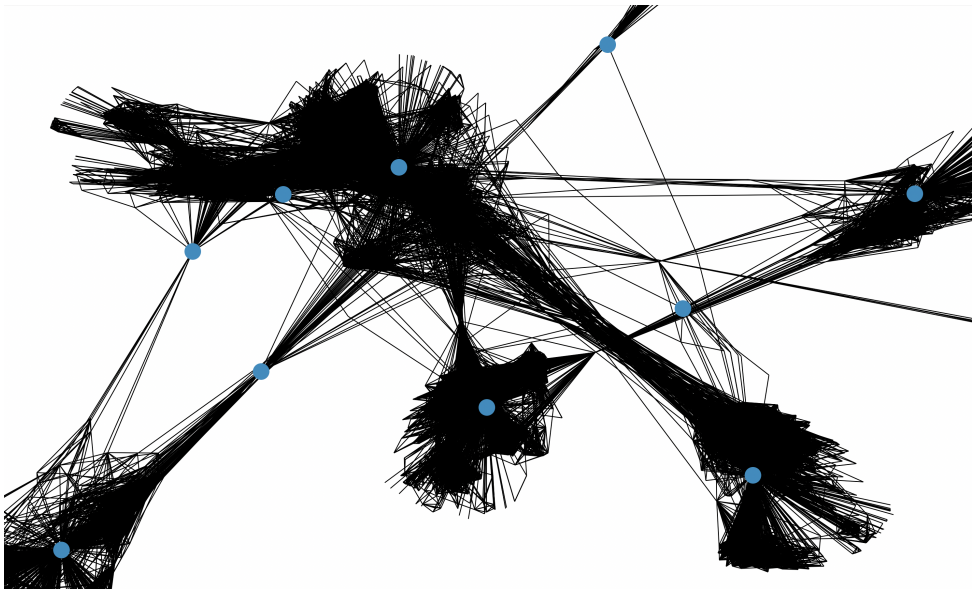

Figure 1: Top 10 Nodes for Betweenness Centrality

Task 2: PageRank

PageRank, invented (ca. 1998) by Larry Page and Sergey Brin was used in the early stages of prototyping the search engine we all rely on daily; Google. The algorithm itself is a estimate to measure a page's importance (or popularity), given it's interconnection to other pages on the web. Phrased simply, a page has more incoming links is more important than one with less incoming links, and a page with a link from an important page is transitively important.

The original paper [2] combined with the method mentioned by Dr. Yin in the tutorial slides gives a good baseline to start implementing the PageRank in your favourite programming language.

Algorithm 1: Power Iteration

1 Set $c^{(0)}$ $i \leftarrow 1$, k $i \leftarrow 1$
2 $c^{(k)}$ $i \leftarrow \alpha A D^{-1} c^{(k-1)} + \beta 1$
3 If $||c^{(k)} - c^{(k-1)}|| > \epsilon$:
4 k $\leftarrow k + 1$, goto **2**

Here we have α being the dampening factor, and β is often set as $1 - \alpha$. The vector holding the ranking of all the nodes is $c^{(k)}$, where k is the current iteration. A note to be made is that $A^T = A$ when we are dealing with an undirected graph, and A is here the adjacency matrix. D is the inverted degree matrix. So the PageRank algorithm is basically recalculating the PageRank vector $c^{(k)}$ until the convergence, where ϵ is the threshold. To have it scale with your dataset, one could multiply ϵ with N, being the size of the dataset.

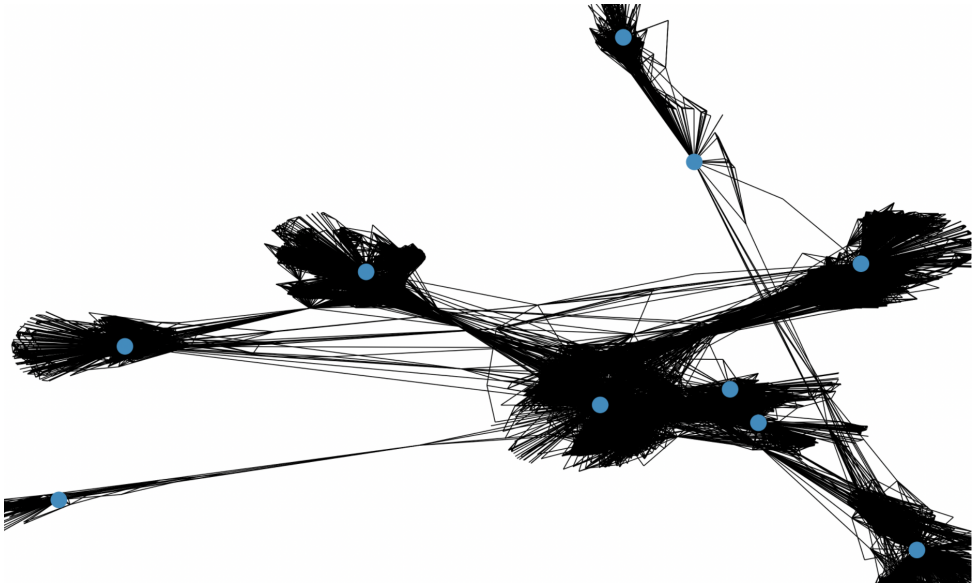

Figure 2: Top 10 PageRank Nodes in Graph

In Figure 4 above we see the top 10 nodes, based on PageRank, visible as blue dots in the populous network of nodes. The black lines are naturally the edges (links) between the top 10 nodes and the rest of the nodes.

The top 10 nodes with their respective PageRank value can be seen below.

Rank	Node	C_p
1	3437	0.007614
2	107	0.006936
3	1684	0.006367
4	0	0.006289
5	1912	0.003876
6	348	0.002348
7	686	0.002219
8	3980	0.002170
9	414	0.001800
10	428	0.001317

Table 2: Results using PageRank Algorithm

Dangling nodes is a node with no outgoing edges (links). It has not been accounted for in this solution, since there were none in the network provided. If one were to tackle it, an approach could be to evenly distribute the page's (node) votes to every other pages (nodes).

Summary

During this assignment I have familiarized myself with graph centrality measures, specifically betweenness centrality and PageRank, which is a variant of a eigenvector centrality measure. The algorithms' pseudocode were clear to follow, and therefore rather straightforward to implement in Python. NetworkX is also of tremendous aid when working with graphs and networks (social networks in this case), and made the implementation without a doubt more smooth.

The Python code for Brandes[1] and PageRank[2] can be found in the Appendix[A].

References

[1] Ulrik Brandes. “A faster algorithm for betweenness centrality”. In: *The Journal of Mathematical Sociology* 25.2 (2001), pp. 163–177. DOI: 10.1080/0022250X.2001.9990249. eprint: https://doi.org/10.1080/0022250X.2001.9990249. URL: https://doi.org/10.1080/0022250X.2001.9990249.

[2] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[3] *NetworkX*. URL: https://networkx.github.io/documentation/stable/ (visited on 04/18/2020).

A Appendix: Code

```python
def brandes_algorithm(vertices, neighbors):
    """
    Ulrik Brandes (2001) A faster algorithm for betweenness centrality , Journal of
    Mathematical Sociology, 25:2, 163-177, DOI: 10.1080/0022250X.2001.9990249
    https://doi.org/10.1080/0022250X.2001.9990249
    """
    C_b = dict((v, 0) for v in vertices) # same as {v: 0 for v in vertices}
    for s in vertices:
        S = deque()  # Stack --> use pop()
        P = dict((w, []) for w in vertices)  # predecessors
        sigma = dict((t, 0) for t in vertices)  # number of shortest paths
        sigma[s] = 1
        delta = dict((t, -1) for t in vertices)
        delta[s] = 0
        Q = deque()  # Queue --> use popleft()
        Q.append(s)
        while Q:
            v = Q.popleft()
            S.append(v)
            for w in neighbors[v]:
                if delta[w] < 0:
                    Q.append(w)
                    delta[w] = delta[v] + 1
                if delta[w] == delta[v] + 1:
                    sigma[w] += sigma[v]
                    P[w].append(v)

        # S returns vertices in order of non-increasing distance from s
        dependency = dict((v, 0) for v in vertices)
        while S:
            w = S.pop()
            for v in P[w]:
                dependency[v] += (sigma[v]/sigma[w]) * (1 + dependency[w])
            if w != s:
                C_b[w] += dependency[w]
    return C_b
```

Listing 1: Brandes Algorithm in Python

```python
def page_rank(G, alpha=0.85, epsilon=1e-4, max_iter=100):
    """ PageRank on undirected graph with L2 norm, assuming no dangling nodes
    Original paper: http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf
    Helpful paper: http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf
    Args:
        G (networkx.classes.graph.Graph): Collection of nodes and edges
        alpha (float): dampening factor
        epsilon (float): threshold value for convergence
        max_iter (int): maximum allowed number of iterations
    Returns:
        A dict containing nodes and their respective PageRank
        {0: 24.680865117351715, ..., 4038: 1.1616657299654847}
    """
    N = G.number_of_nodes()
    A = G.adj
    D = dict((n, 1/d) for n, d in G.degree())
    c = dict((n, 1.0) for n in G)
    for _ in range(max_iter):
        prev_c = c
        c = dict((n, 0.0) for n in G)
        for node in A:
            for nbr in A[node]:
                c[nbr] += alpha * D[node] * prev_c[node]
            c[node] += (1 - alpha)
        delta = (np.linalg.norm([c[n] - prev_c[n] for n in c]))
        if delta < N * epsilon:
            return c
    print(f"Did not converge in {max_iter} iterations.")
```

Listing 2: PageRank Algorithm in Python