PageRank algorithm

Abhijeet Singh 2014B3A70596H
Karran Pandey 2014B4A70648H

Dataset

This dataset consists of 'circles' (or 'lists') from Twitter. Twitter data was crawled from public sources. The dataset includes node features (profiles), circles, and ego networks. The dataset used consists of a two tuple (A,B) where A and B corresponded to the ids of the individuals A and B. (A,B) indicates A follows B.

Language used: Python 3

Data Structures used

- A dictionary of lists was used to store the inlinks for a particular node.
- Another dictionary was used to store the number of outlinks for one particular node.
- Two dictionaries were used to convert from index to id and from id to index respectively.

Python modules used

- networkx (plotting the directed graph)
- seaborn (plotting)
- matplotlib (plotting)
- numpy (minor calculations)

Enhancements incorporated

Since it didn't make a lot of sense to implement trust rank on the current dataset, given that it had unknown ids. The following enhancements were incorporated. Adjacency list implementation was used which saved on space.

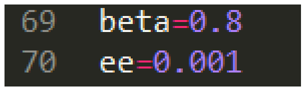
1. Adjacency list implementation for very large graph.
2. Also each iteration was visualized and the change in 20 random nodes was observed graphically

Formula used

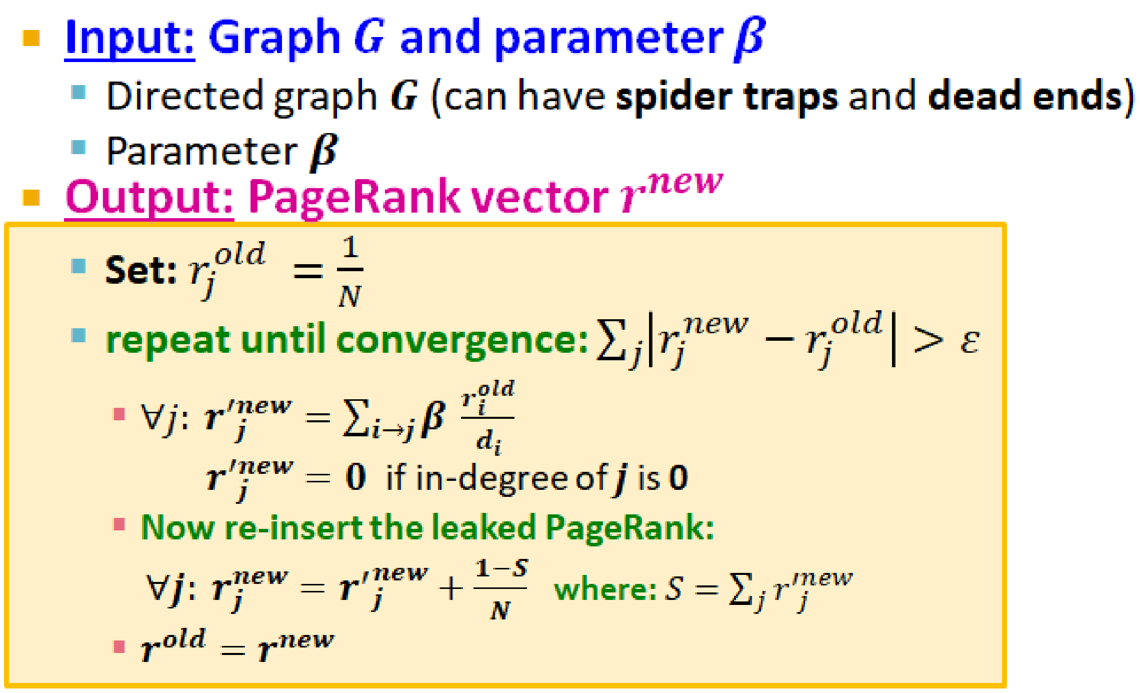
The power iteration

$$\mathbf{r = M.r}$$

was used in principle. However, since spider traps and dead ends had to be handled, the version satisfying these conditions was used.
Values of e and Beta used were:



```
beta=0.8
ee=0.001
```



- Input: Graph G and parameter β
  - Directed graph G (can have **spider traps** and **dead ends**)
  - Parameter β
- Output: PageRank vector r^{new}
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left| r_j^{new} - r_j^{old} \right| > \varepsilon$
    - $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$
      $r'^{new}_j = 0$ if in-degree of j is 0
    - Now re-insert the leaked PageRank:
      $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-S}{N}$ where: $S = \sum_j r'^{new}_j$
    - $r^{old} = r^{new}$

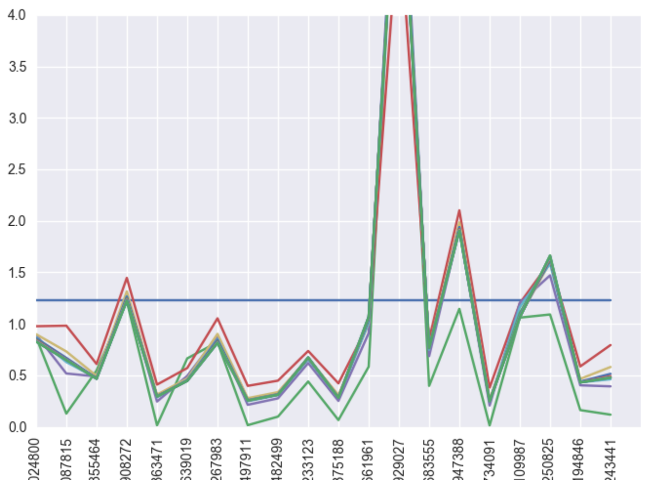
Running time

Since the graph was pretty big, having 83106 nodes and 242000 edges, preprocessing took considerable amount of time. The in_links dictionary (of lists) which stored the in links to a particular node took runtime O(edges*log(max_inlinks)*max_links) where max_links is the number of maximum in links to any of the nodes in the entire list.

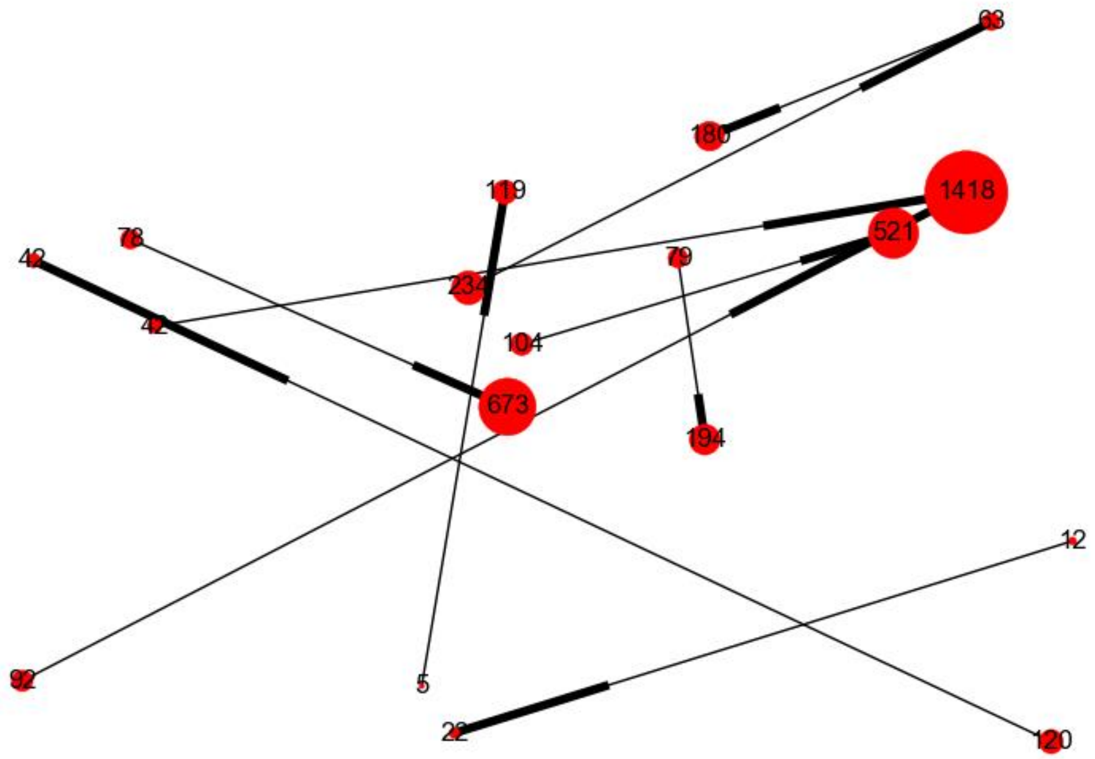
For each iteration of the power iteration, The complexity was O(Vertexes*(max_inlinks+log(Vertexes)).

Problems with PageRank

- Susceptible to Link Spam
- Uses in links as a measure of importance and nothing else. In a social media setting though, the number of followers is a huge factor in determining the influencers.



Dynamic graph showing the r values for 20 random ids



Directed graph showing the page rank formulation for first 10 edges (magnified by 10^5)